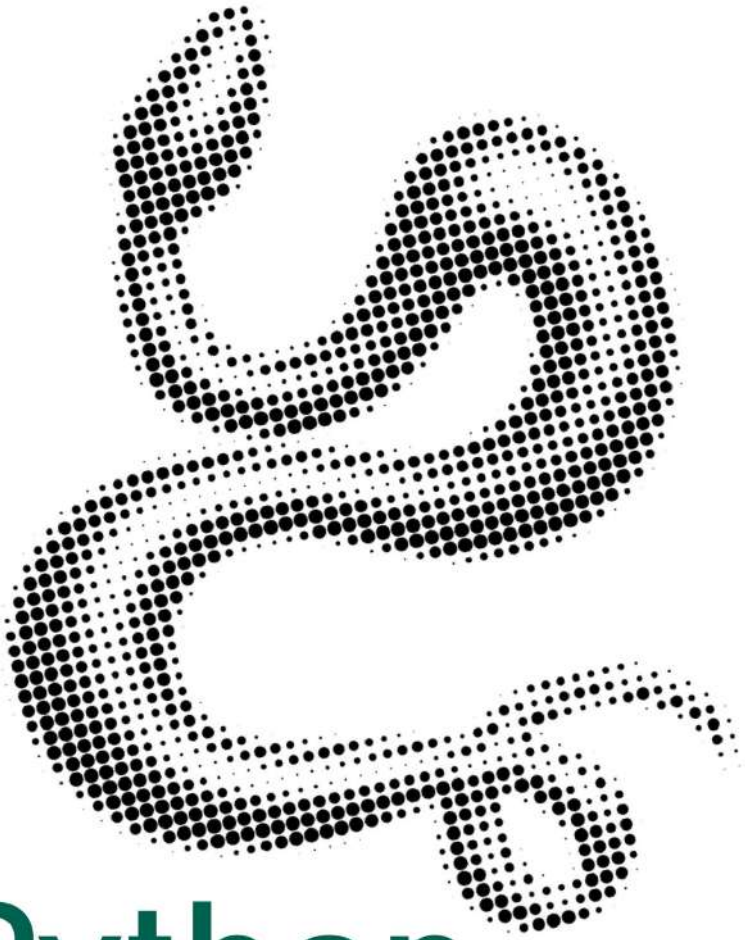
Мэттью Уайтсайд



Python в задачах и упражнениях

Мэттью Уайтсайд

Python
в задачах и упражнениях

Matthew Whiteside

Programming Puzzles

Python Edition

Мэттью Уайтсайд

Python в задачах и упражнениях

2025

Уайтсайд М.
Python в задачах и упражнениях – 2025. - 214 с.

Задачи, собранные в этой книге, задуманы так, чтобы читатели любого уровня подготовки могли испытать на них свои силы. Первая часть книги состоит из серьезных задач, подобранных по нарастанию сложности. Рекомендуется решать их по порядку, ничего не пропуская. Во второй части представлены шуточные задачи, требующие твор-ческого мышления и умения использовать незнакомые библиотеки.

Ожидается, что читатель владеет основами Python и знаком с такими базовыми понятиями, как переменные, условные предложения, циклы и функции.

Содержание

Приступая к работе 12

Введение 13

Подготовка среды 14

Нотация О большое 16

И еще несколько предварительных замечаний 18

Серьезные задачи 20

Задача 1 21
Задача 2 22
Задача 3 23
Задача 4 24
Задача 5 25
Задача 6 26
Задача 7 27
Задача 8 28
Задача 9 29
Задача 10 31
Задача 11 32
Задача 12 33
Задача 12.1 (дополнительная) 34
Задача 13 35
Задача 14 36
Задача 15 37
Задача 16 38
Задача 17 39
Задача 18 40
Задача 19 41
Задача 20 43
Задача 21 44

Задача 22....45
Задача 23....46
Задача 24....48
Задача 25....50
Задача 26....51
Задача 27....52
Задача 28....54
Задача 29....56
Задача 30....59
Задача 31....60
Задача 31.1 (дополнительная)....61
Задача 32....62
Задача 33....63
Задача 34....64
Задача 35....65
Задача 36....66
Задача 37....68
Задача 38....69
Задача 38.1 (дополнительная)....70
Задача 39....71
Задача 40....73
Задача 41....74
Задача 41.1 (дополнительная)....76
Задача 42....78
Задача 43....79
Задача 44....81
Задача 45....83
Задача 45.1 (дополнительная)....85
Задача 46....86
Задача 47....87
Задача 48....89
Задача 48.1 (дополнительная)....91
Задача 49....93
Задача 49.1 (дополнительная)....94
Задача 49.2 (дополнительная)....95
Задача 50....96

Шуточные задачи....98
Обратный поиск в DNS....99
Матрешки....101

Фрактальное дерево 103
Пинг-понг 104
Средство рисования 105

Серьезные задачи: указания 106

Задача 1 106
Задача 2 106
Задача 3 107
Задача 4 107
Задача 5 108
Задача 6 108
Задача 7 108
Задача 8 109
Задача 9 109
Задача 10 109
Задача 11 110
Задачи 12 и 12.1 110
Задача 13 110
Задача 14 111
Задача 15 111
Задача 16 111
Задача 17 112
Задача 18 113
Задача 19 115
Задача 20 115
Задача 21 116
Задача 22 117
Задача 23 117
Задача 24 117
Задача 25 118
Задача 26 118
Задача 27 119
Задача 28 119
Задача 29 120
Задача 30 120
Задача 31 120
Задача 31.1 (дополнительная) 121
Задача 32 122
Задача 33 122
Задача 34 123

Задача 35....123
Задача 36....124
Задача 37....125
Задача 38....125
Задача 38.1 (дополнительная)....126
Задача 39....127
Задача 40....127
Задача 41....127
Задача 41.1 (дополнительная)....128
Задача 42....129
Задача 43....130
Задача 44....131
Задача 45....132
Задача 45.1 (дополнительная)....133
Задача 46....134
Задача 47....135
Задача 48....136
Задача 48.1 (дополнительная)....136
Задача 49....137
Задача 49.1 (дополнительная)....137
Задача 49.2 (дополнительная)....138
Задача 50....138

Серьезные задачи: решения....139

Задача 1....139
Задача 2....140
Задача 3....140
Задача 4....141
Задача 5....141
Задача 6....141
Задача 7....142
Задача 8....142
Задача 9....143
Задача 10....144
Задача 11....144
Задача 12....145
Задача 12.1 (дополнительная)....145
Задача 13....145
Задача 14....146
Задача 15....146

Задача 16.....147
Задача 17.....148
Задача 18.....148
Задача 19.....149
Задача 20.....150
Задача 21.....150
Задача 22.....151
Задача 23.....151
Задача 24.....152
Задача 25.....153
Задача 26.....154
Задача 27.....155
Задача 28.....156
Задача 29.....157
Задача 30.....159
Задача 31.....160
Задача 31.1 (дополнительная).....161
Задача 31.1 – другое решение.....162
Задача 32.....163
Задача 33.....164
Задача 34.....165
Задача 35.....167
Задача 36.....169
Задача 37.....171
Задача 38.....172
Задача 38.1 (дополнительная).....173
Задача 39.....174
Задача 40.....175
Задача 41.....176
Задача 41.1 (дополнительная).....177
Задача 42.....179
Задача 43.....180
Задача 44.....181
Задача 45.....183
Задача 45.1 (дополнительная).....184
Задача 46.....187
Задача 47.....188
Задача 48.....190
Задача 48.1 (дополнительная).....191
Задача 49.....193
Задача 49.1 (дополнительная).....195

Задача 49.2 (дополнительная) 196
Задача 50 197

Шуточные задачи: решения 199
Обратный поиск в DNS 199
Матрешки 200
Фрактальное дерево 202
Пинг-понг 204
Пинг-понг (дополнение) 206
Средство рисования 208
Средство рисования (дополнение) 210

Приступая к работе

Добро пожаловать! Прежде чем с головой окунуться в мир задач на Python, остановимся на нескольких важных моментах, чтобы в дальнейшем вы получали от книги ничем не омраченное удовольствие.

Введение

Задачи, собранные в этой книге, задуманы так, чтобы читатели любого уровня могли испытать на них свои силы. Некоторые задачи предназначены для тех, кто только начинает свой путь в мир Python, другие, потруднее, ориентированы на более опытных программистов.

Книга состоит из двух частей: серьезные задачи и шуточные задачи. В первой части 50 (+ несколько дополнительных) задач, трудность которых постепенно растет. Это основная часть книги, рекомендуется начать с задачи 1 и дальше решать по порядку, ничего не пропуская.

В шуточных задачах требуется проявить творческие способности и воспользоваться несколькими предлагаемыми Python библиотеками. Их рекомендуется попробовать, когда вы устанете от серьезных задач и захотите передохнуть, хотя легкими их все же не назовешь!

Примечание: для решения задач в этой книге необходимо владеть хотя бы основами Python. Знакомство с такими базовыми понятиями, как переменные, условные предложения, циклы и функции, сильно поможет в решении задач.

Подготовка среды

Прежде чем решать задачи, нужно подготовить рабочую среду, иначе вы не сможете проверить свои решения! По счастью, подготовить среду для Python совсем несложно.

Установка Python

Процедура установки Python зависит от операционной системы, но в общем и целом состоит из следующих шагов:

1. Зайти на официальный сайт Python по адресу *python.org*.
2. Перейти в раздел «Downloads».
3. Выбрать подходящий для своей операционной системы (Windows, macOS или Linux) установщик и **версию Python не ниже 3.10**.
4. Скачать установщик и следовать его инструкциям. Если установщик спросит, хотите ли вы установить менеджер пакетов pip, соглашайтесь.

По завершении установки можете открыть командную строку или терминал и ввести команду *python –version*, чтобы убедиться, что Python установлен правильно. Эта команда должна напечатать номер установленной версии. Если возникли проблемы, можете попробовать команду *python3 --version* или *python3.10 --version*.

Редактор кода

Вам также понадобится редактор кода, чтобы вводить и выполнять код на Python. Вариантов много, ниже перечислены некоторые наиболее популярные:

- IDLE – входит в комплект поставки Python;
- Visual Studio Code – *code.visualstudio.com*;
- PyCharm – *jetbrains.com/pycharm*.

Для этой книги подойдет любой из вышеперечисленных редакторов, но если вы решите продолжить знакомство с Python, то име-

ет смысл перейти на более мощную IDE, такую как Visual Studio Code или PyCharm.

Внешние библиотеки Python

В нескольких задачах используются внешние библиотеки Python. Их можно установить с помощью менеджера Python-пакетов pip.

Проверить, установлен ли на вашей машине pip, позволит команда `python -m pip -version`. Следует использовать то же название программы, что и выше: если сработала команда *python3 –version*, то и для проверки наличия pip нужно набирать *python3 –m pip --version*.

Подробнее о pip можно прочитать на странице *docs.python.org/3/installing/index.html*.

Используемые в книге внешние модули можно установить, выполнив следующие команды:

- *python -m pip install pygame*;
- *python –m pip install PythonTurtle*.

Git (факультативно)

К этой книге прилагается git-репозиторий, в котором имеются заготовки кода и решения всех задач. Все это можно скачать непосредственно с GitHub, но правильнее будет клонировать репозиторий с помощью программы git. Если вы знакомы с git, то можете просто выполнить следующие команды и затем пропустить этот раздел:

- HTTPS: *git clone https://github.com/MatWhiteside/python-puzzlebook.git*;
- SSH: *git clone git@github.com:MatWhiteside/python-puzzlebook.git*.

Если же вы незнакомы с git, то сначала установите ее на свой компьютер. Я не стану здесь описывать эту процедуру, потому что она зависит от операционной системы, но в сети достаточно руководств, которые помогут вам это сделать.

Если вы не сумеете заставить git работать, тоже не беда. Перейдите по ссылке на GitHub в браузере и скачайте код в виде zip-файла. Затем можете распаковать его в удобный вам каталог и приступать к работе. Ссылка такая: *github.com/MatWhiteside/python-puzzle-book*.

Нотация O большое

В книге есть несколько более трудных задач, в которых упоминается временная или пространственная сложность. Например, требование может звучать так: «ваше решение должно иметь временную сложность не более $O(n)$». Но что это значит?

Нотация *O большое* часто используется, когда нужно описать сложность проблемы, и определяет сложность фрагмента кода в худшем случае. Выглядит она так: $O(...)$.

Примеры

$O(1)$ (постоянное время) означает, что выполнение кода всегда занимает одно и то же время вне зависимости от входных данных. Вот пример функции со сложностью $O(1)$:

```
def add_five(input_num):
    return input_num +_ 5
```

Не важно, равно `input_num` нулю или 99999, для сложения придется выполнить только одну строку кода. Поэтому сложность функции постоянна.

Запись $O(n)$ означает, что сложность линейно возрастает с ростом размера входных данных, представленного переменной `n`. Вот пример функции с линейной сложностью:

```
def print_list(input_list):
    for item in input_list:
        print(item)
```

Если длина списка `input_list` равна 5, то `print(item)` будет выполнено 5 раз. Если же длина равна 9999, то `print(item)` будет выполнено 9999 раз. Это и означает, что функция имеет линейную временную сложность.

Запись $O(n^2)$ означает, что временная сложность растет как квадрат размера входных данных. Например:

```
def print_list_nested(input_list):
    for item in input_list:
```

```
        for inner_item in input_list:
            print(item, " + ", inner_item)
```

Если длина `input_list` равна 5, то предложение `print(item, " + ", inner_item)` будет выполнено 25 раз. Если же длина `input_list` равна 1000, то оно будет выполнено 1 000 000 раз.

Если вы запутались, не переживайте. Упоминаний о нотации «О большое» не будет еще довольно долго, а если, когда мы до них доберемся, вам еще понадобится помощь, то вы сможете найти достаточно ресурсов в сети. Просто поищите по запросу «Big O notation explained».

И еще несколько предварительных замечаний

Надеюсь, что вы уже готовы приступить к решению задач! Но прежде еще несколько моментов...

1. Не страшно, если ваши решения сбоят на неправильных входных данных. Мы здесь не пишем производственный код, и если в условии сказано, что на вход подается список целых чисел, то можете предполагать, что так и будет. Но все же не забывайте рассматривать крайние случаи, например что будет, если список пуст? Или если входное число отрицательно?
2. Для всех серьезных задач имеются заготовки кода, помогающие вам быстрее начать работу. В большинстве случаев определена только одна функция, но это не значит, что вы не можете определить дополнительные. Это всего лишь подсказка, вы можете вообще не использовать заготовки – дело ваше!
3. В ряде задач имеются дополнительные условия, которым нужно удовлетворить, чтобы задача считалась решенной правильно. Если возникли затруднения, попробуйте решить задачи без учета дополнительных условий, а затем подумайте, как улучшить решение.
4. Всюду в коде используются типизированные определения функций, чтобы было понятно, каковы типы входов и выходов функции. Если вы незнакомы с типизацией в Python, не расстраивайтесь: с вашей точки зрения, она ничего не меняет. Можете даже вообще удалить типизацию из определений функций и работать, как вы привыкли.

Пример типизированной функции:

```
def filter_strings_containing_a(input_strs: list[str])
          -> list[str]:
```

Та же функция без типизации:

```
def filter_strings_containing_a(input_strs):
```

Серьезные задачи

Задачи в этой части книги начинаются с легких, но постепенно их трудность возрастает.

В каждой задаче формулируется, что нужно сделать, а от вас требуется написать соответствующую функцию. Чтобы вы могли сосредоточиться на важных аспектах задачи, приводится также заготовка кода, которая поможет вам сразу перейти к кодированию.

Если задача никак не решается, то далее в книге имеется раздел с указаниями. Я настоятельно рекомендую воспользоваться указаниями, прежде чем заглядывать в решение. Лучший способ чему-то научиться – помучиться над задачей и все же решить ее самостоятельно.

Важно: не импортируйте никакие библиотеки, если в задаче это явно не требуется!

Задача 1

Задание

Определите функцию *filter_strings_containing_a*, принимающую один параметр:

Имя	Тип	Пример входа
`input_strs`	список строк	`["apple", "banana", "cherry", "date"]`

Функция должна возвращать новый список, содержащий только строки, содержащие букву «a».

Заготовка кода

```
def filter_strings_containing_a(input_strs: list[str])
    -> list[str]:
    # Здесь должна быть ваша реализация

print(filter_strings_containing_a(
    ["apple", "banana", "cherry", "date"])
)
```

Примеры

```
Вход:   ["apple", "banana", "cherry", "date"]
Выход:  ["apple", "banana", "date"]

Вход:   []
Выход:  []

Вход:   ["bbbb", "cccc"]
Выход:  []
```

Задача 2

Задание

Определите функцию *sum_if_less_than_fifty*, принимающую два параметра:

Имя	Тип	Пример входа
num_one	int	20
num_two	Int	25

Функция должна возвращать:

- сумму двух чисел, если эта сумма меньше 50;
- None, если сумма двух чисел больше или равна 50.

Заготовка кода

```
def sum_if_less_than_fifty(num_one: int, num_two: int)
    -> int | None:
    # Здесь должна быть ваша реализация

print(sum_if_less_than_fifty(20, 20))
print(sum_if_less_than_fifty(20, 30))
```

Примеры

```
Входы:
 - num_one = 20
 - num_two = 20
Выход: 40
Входы:
 - num_one = 20
 - num_two = 30
Выход: None
Входы:
 - num_one = 20
 - num_two = 100
Выход: None
```

Задача 3

Задание

Определите функцию *sum_even*, принимающую один параметр:

Имя	Тип	Пример входа
`input_nums`	список `int`	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Функция должна возвращать сумму всех четных чисел в списке:

Заготовка кода

```
def sum_even(input_nums: list[int]) -> int:
    # Здесь должна быть ваша реализация

print(sum_even([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
print(sum_even([10, 20, 30, 40, 50]))
print(sum_even([9, 7, 5, 3, 1]))
```

Примеры

```
Вход:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Выход: 30

Вход:  [10, 20, 30, 40, 50]
Выход: 150

Вход:  [[9, 7, 5, 3, 1]]
Выход: 0
```

Задача 4

Задание

Определите функцию *remove_vowels*, принимающую один параметр:

Имя	Тип	Пример входа
`input_str`	`str`	«Hello, World!»

Функция должна возвращать новую строку, из которой удалены все гласные.

Заготовка кода

```
def remove_vowels(input_str: str) -> str:
    # Здесь должна быть ваша реализация

print(remove_vowels("Hello, World!"))
print(remove_vowels("aeiouAEIOU"))
print(remove_vowels("zzxxxccvvvbbnnmmmLLKKJJHH"))
```

Примеры

```
Вход:  "Hello, World!"
Выход: "Hll, Wrld!"

Вход:  "aeiouAEIOU"
Выход: ""

Вход:  "zzxxxccvvvbbnnmmmLLKKJJHH"
Выход: "zzxxxccvvvbbnnmmmLLKKJJHH"
```

Задача 5

Задание

Определите функцию *get_longest_string*, принимающую один параметр:

Имя	Тип	Пример входа
`input_strs`	список строк	[«cat», «dog», «bird», «lizard»]

Функция должна возвращать самую длинную строку в списке. Если таких несколько, то должна быть возвращена та, что встречается в списке первой.

Заготовка кода

```
def get_longest_string(input_strs: list[str]) -> str:
    # Здесь должна быть ваша реализация

print(get_longest_string(["cat", "dog", "bird", "lizard"]))
print(get_longest_string(["cat", "dog", "bird", "wolf"]))
print(get_longest_string(["a", "b", "c", "d"]))
```

Примеры

```
Вход:  ["cat", "dog", "bird", "lizard"]
Выход: "lizard"

Вход:  ["cat", "dog", "bird", "wolf"]
Выход: "bird"

Вход:  ["a", "b", "c", "d"]
Выход: "a"
```

Задача 6

Задание

Определите функцию *filter_even_length_strings*, принимающую один параметр:

Имя	Тип	Пример входа
`input_strs`	список строк	[«cat», «dog», «fish», «elephant»]

Функция должна возвращать новый список, в котором оставлены только строки, содержащие четное число символов.

Заготовка кода

```
def filter_even_length_strings(input_strs: list[str]) -> list[str]:
    # Здесь должна быть ваша реализация

print(filter_even_length_strings(["cat", "dog", "fish", "elephant"]))
print(filter_even_length_strings(["q", "w", "e", "r", "t", "y"]))
print(filter_even_length_strings(["qq", "ww", "ee", "rr", "t", "yy"]))
```

Примеры

```
Вход:  ["cat", "dog", "fish", "elephant"]
Выход: ["fish", "elephant"]

Вход:  ["q", "w", "e", "r", "t", "y"]
Выход: []

Вход:  ["qq", "ww", "ee", "rr", "t", "yy"]
Выход: ["qq", "ww", "ee", "rr", "t", "yy"]
```

Задача 7

Задание

Определите функцию *reverse_elements*, принимающую один параметр:

Имя	Тип	Пример входа
`input_nums`	список `int`	[1, 2, 3, 4, 5]

Функция должна возвращать новый список, в котором порядок элементов исходного списка изменен на противоположный.

Заготовка кода

```
def reverse_elements(input_nums: list[int]) -> list[int]:
    # Здесь должна быть ваша реализация

print(reverse_elements([1, 2, 3, 4, 5]))
print(reverse_elements([]))
print(reverse_elements([20, 15, 25, 10, 30, 5, 0]))
```

Примеры

```
Вход:  [1, 2, 3, 4, 5]
Выход: [5, 4, 3, 2, 1]

Вход:  []
Выход: []

Вход:  [20, 15, 25, 10, 30, 5, 0]
Выход: [0, 5, 30, 10, 25, 15, 20]
```

Задача 8

Задание

Определите функцию *filter_type_str*, принимающую один параметр:

Имя	Тип	Пример входа
`input_list`	список строк или `int`	[«hello”, 1, 2, “www”]

Функция должна возвращать новый список, содержащий только строки из оригинального списка.

Заготовка кода

```
def filter_type_str(input_list: list[str | int]) -> list[str]:
    # Здесь должна быть ваша реализация

print(filter_type_str(["hello", 1, 2, "www"]))
print(filter_type_str([]))
print(filter_type_str([1, 2, 3, 4, 5]))
```

Примеры

```
Вход:  ["hello", 1, 2, "www"]
Выход: ["hello", "www"]

Вход:  []
Выход: []

Вход:  [1, 2, 3, 4, 5]
Выход: []
```

Задача 9

Задание

Определите функцию *string_to_morse_code*, принимающую один параметр:

Имя	Тип	Пример входа
input_str	str	«HELLO, WORLD!»

Функция должна возвращать код Морзе, эквивалентный входной строке. Функция должна удовлетворять следующим требованиям:

- код Морзе «точка» должен быть представлен точкой;
- код Морзе «тире» должен быть представлен дефисом;
- между любыми двумя буквами в коде Морзе должен быть пробел, например «.-(пробел)-...»;
- функция должна поддерживать следующие символы во входной строке:
  - цифры и буквы, строчные и заглавные;
  - специальные символы (, . : ? ‘ - / ( ) “ @ = + !);
- если во входной строке встречается пробел, то в выходной он должен быть представлен косой чертой.

Заготовка кода

```
def string_to_morse_code(input_str: str) -> str:
    morse_dict = {"a": ".-",     "b": "-...",   "c": "-.-.",   "d": "-..",
                  "e": ".",      "f": "..-.",   "g": "--.",    "h": "....",
                  "i": "..",     "j": ".---",   "k": "-.-",    "l": ".-..",
                  "m": "--",     "n": "-.",     "o": "---",    "p": ".--.",
                  "q": "--.-",   "r": ".-.",    "s": "...",    "t": "-",
                  "u": "..-",    "v": "...-",   "w": ".--",    "x": "-..-",
                  "y": "-.--",   "z": "--..",   "0": "-----",  "1": ".----",
                  "2": "..---",  "3": "...--",  "4": "....-",  "5": ".....",
                  "6": "-....",  "7": "--...",  "8": "---..",  "9": "----.",
                  ",": "--..--", ".": ".-.-.-", ":": "---...", "?": "..--..",
                  "'": ".----.", "-": "-....-", "/": "-..-.",  "(": "-.--.",
                  ")": "-.--.-", '"': ".-..-.", "@": ".--.-.", "=": "-...-",
                  "+": ".-.-.",  "!": "-.-.--"}

    # Здесь должна быть ваша реализация

print(string_to_morse_code("HELLO, WORLD!"))
print(string_to_morse_code("abcdefghijklmnopqrstuvwxyz,.:?'-/()\"@=+!"))
print(string_to_morse_code(""))
```

Примеры

```
Вход:  "HELLO, WORLD!"
Выход: ".... . .-.. .-.. --- --..-- . .-- --- .-. .-.. -.. -.-.-"

Вход:  "abcdefghijklmnopqrstuvwxyz,.:?'-/()\"@=+!"
Выход: ".- -... -.-. -.. . ..-. --. .... .. .--- -.- .-.. -- -. --
- .--. --.- .-. ... - ..- ...- .-- -..- -.-- --.. --..-- .-.-.- ---
... ..--.. .----. -....- -..-. -.--. -.--.- .-..-. .--.-. -...- .-
.-. -.--"

Вход:  ""
Выход: ""
```

Задача 10

Задание

Определите функцию *get_second_largest_number*, принимающую один параметр:

Имя	Тип	Пример входа
`input_nums`	список `int`	[1, 2, 3, 4, 5]

Функция должна возвращать второе по величине число в списке. Если второго по величине числа нет, то функция должна возвращать `None`.

Заготовка кода

```
def get_second_largest_number(input_nums: list[int]) -> int | None:
    # Здесь должна быть ваша реализация

print(get_second_largest_number([1, 2, 3, 4, 5]))
print(get_second_largest_number([3, 45, 345, 435, 345, 43, 56, 34, 234, 34]))
print(get_second_largest_number([1]))
```

Примеры

```
Вход:  [1, 2, 3, 4, 5]
Выход: 4

Вход:  [3, 45, 345, 435, 345, 43, 56, 34, 234, 34]
Выход: 345

Вход:  [1]
Выход: None
```

Задача 11

Задание

Определите функцию *format_number_with_commas*, принимающую один параметр:

Имя	Тип	Пример входа
`input_num`	`int`	1000000

Функция должна возвращать строковое представление числа, в котором группы по 3 разряда (начиная справа) разделены запятыми.

Заготовка кода

```
def format_number_with_commas(input_num: int) -> str:
    # Здесь должна быть ваша реализация

print(format_number_with_commas(1000000))
print(format_number_with_commas(12345))
print(format_number_with_commas(-99999999))
```

Примеры

```
Вход:  1000000
Выход: "1,000,000"

Вход:  12345
Выход: "12,345"

Вход:  -99999999
Выход: "-99,999,999"
```

Задача 12

Предварительные сведения

ASCII – стандартная кодировка данных для передачи между компьютерами по каналам связи. ASCII сопоставляет числовые значения буквам, цифрам, знакам препинания и другим символам.

Для преобразования символа в ASCII-код мы можем обратиться к таблице ASCII по адресу *https://www.asciitable.com*.

В этой таблице пять столбцов: Dec, Hx, Oct, Html и Chr. Для преобразования символа в эквивалентный ASCII-код мы находим его в столбце Chr и берем соответствующее число из столбца Dec. Например, символу “A” соответствует код 65.

Возможно и обратное преобразование: если найти в столбце Dec ASCII-код 65, то из столбца Chr мы увидим, что ему соответствует символ «A».

Задание

Определите функцию *string_to_ascii*, принимающую один параметр:

Имя	Тип	Пример входа
`input_str`	`str`	«Programming puzzles!»

Функция должна возвращать список, содержащий числовые ASCII-коды всех символов строки.

Заготовка кода

```
def string_to_ascii(input_str: str) -> list[int]:
    # Здесь должна быть ваша реализация

print(string_to_ascii("Programming puzzles!"))
print(string_to_ascii(""))
print(string_to_ascii("aA"))
```

Примеры

```
Вход:  "Programming puzzles!"
Выход: [80, 114, 111, 103, 114, 97, 109,1 09, 105, 110, 103, 32,
112, 117, 122, 122, 108, 101, 115, 33]

Вход:  ""
Выход: []

Вход: "aA"
Выход: [97, 65]
```

Задача 12.1 (дополнительная)

Задание

Определите функцию *ascii_to_string*, принимающую один параметр:

Имя	Тип	Пример входа
`input_ascii_codes`	список `int`	[80, 114, 111, 103, 114, 97, 109, 109, 105, 110, 103, 32, 112, 117, 122, 122, 108, 101, 115, 33]

Функция должна возвращать строку, содержащую символы с указанными в списке ASCII-кодами.

Заготовка кода

```
def ascii_to_string(input_ascii_codes: list[int]) -> str:
    # Здесь должна быть ваша реализация

print(ascii_to_string([80, 114, 111, 103, 114, 97, 109, 109, 105,
110, 103, 32, 112, 117, 122, 122, 108, 101, 115, 33]))
print(ascii_to_string([]))
print(ascii_to_string([97, 65]))
```

Примеры

```
Вход:  [80, 114, 111, 103, 114, 97, 109, 109, 105, 110, 103, 32,
112, 117, 122, 122, 108, 101, 115, 33]
Выход: "Programming puzzles!"

Вход: []
Выход:  ""

Вход: [97, 65]
Выход: "aA"
```

Задача 13

Задание

Определите функцию *filter_strings_with_vowels*, принимающую один параметр:

Имя	Тип	Пример входа
`input_strs`	список `str`	`["apple", "banana", "zyxvb"]`

Функция должна возвращать новый список, содержащий строки, в которых есть хотя бы одна гласная.

Заготовка кода

```
def filter_strings_with_vowels(input_strs: list[str]) -> list[str]:
    # Здесь должна быть ваша реализация

print(filter_strings_with_vowels(["apple", "banana", "zyxvb"]))
print(filter_strings_with_vowels([]))
print(filter_strings_with_vowels(["q", "w", "e", "r", "t", "y"]))
```

Примеры

```
Вход:  ["apple", "banana", "zyxvb"]
Выход: ["apple", "banana"]

Вход: []
Выход: []

Вход: ["q", "w", "e", "r", "t", "y"]
Выход: ["e"]
```

Задача 14

Задание

Определите функцию *reverse_first_five_positions*, принимающую один параметр:

Имя	Тип	Пример входа	Ограничение
`input_nums`	список `int`	`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`	`len(input_nums) == 10`

Функция должна возвращать новый список, в котором первые пять элементов оригинального списка переставлены в обратном порядке. В решении следует использовать срезы Python и не использовать циклы.

Заготовка кода

```
def reverse_first_five_positions(input_nums: list[int]) -> list[int]:
    # Здесь должна быть ваша реализация

print(reverse_first_five_positions([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
print(reverse_first_five_positions([100, 90, 80, 70, 60, 50, 40, 30, 20, 10]))
print(reverse_first_five_positions([-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]))
```

Примеры

```
Вход:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Выход: [5, 4, 3, 2, 1, 6, 7, 8, 9, 10]

Вход: [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
Выход: [60, 70, 80, 90, 100, 50, 40, 30, 20, 10]

Вход: [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
Выход: [-5, -4, -3, -2, -1, -6, -7, -8, -9, -10]
```

Задача 15

Задание

Определите функцию *filter_palindromes*, принимающую один параметр:

Имя	Тип	Пример входа
input_strs	список str	["cat", "dog", "racecar", "deified", "giraffe"]

Функция должна возвращать новый список, содержащий только строки, являющиеся палиндромами.

Заготовка кода

```
def filter_palindromes(input_strs: list[str]) -> list[str]:
    # Здесь должна быть ваша реализация

print(filter_palindromes(["cat", "dog", "racecar", "deified", "giraffe"]))
print(filter_palindromes(["kayak", "deified", "rotator", "repaper", "deed",
"a"]))
print(filter_palindromes(["ab", "ba", "cd", "ef", "pt"]))
```

Примеры

```
Вход:  ["cat", "dog", "racecar", "deified", "giraffe"]
Выход: [""racecar", "deified"]

Вход: ["kayak", "deified", "rotator", "repaper", "deed", "a"]
Выход: ["kayak", "deified", "rotator", "repaper", "deed", "a"]

Вход: ["ab", "ba", "cd", "ef", "pt"]
Выход: []
```

Задача 16

Задание

Определите функцию *censor_python*, принимающую один параметр:

Имя	Тип	Пример входа
`input_strs`	список `str`	`["python", "hello", "HELLO"]`

Функция должна возвращать новый список строк, в которых буквы «P», «Y», «T», «H», «O», «N» (в любом регистре) заменены буквой «X».

Заготовка кода

```
def censor_python(input_strs: list[str]) -> list[str]:
    # Здесь должна быть ваша реализация

print(censor_python(["python", "hello", "HELLO"]))
print(censor_python(["abcdefg"]))
print(censor_python([]))
```

Примеры

```
Вход:  ["python", "hello", "HELLO"]
Выход: ["XXXXXX", "HellX", "XELLX"]

Вход: ["abcdefg"]
Выход: ["abcdefg"]

Вход: []
Выход: []
```

Задача 17

Предварительные сведения

Будем называть строку счастливой, если любые три последовательных символа в ней различны.

Примеры счастливых строк:

- «abcdefg»;
- «qwerty»;
- «abcabcabcabc».

Примеры несчастливых строк:

- «aaaaaaaa»;
- «cbc»;
- «hello».

Задание

Определите функцию *check_if_string_is_happy*, принимающую один параметр:

Имя	Тип	Пример входа
`input_str`	`str`	`"abcdefg"`

Функция должна возвращать булево значение, показывающее, является строка счастливой или нет.

Заготовка кода

```
def check_if_string_is_happy(input_str: str) -> bool:
    # Здесь должна быть ваша реализация

print(check_if_string_is_happy("abcdefg"))
print(check_if_string_is_happy("abcabcabcabc"))
print(check_if_string_is_happy("hello"))
```

Примеры

```
Вход:  "abcdefg"
Выход: True

Вход: "abcabcabcabc"
Выход: True

Вход: "hello"
Выход: False
```

Задача 18

Задание

Определите функцию *get_number_of_digits*, принимающую один параметр:

Имя	Тип	Пример входа	Ограничение
`input_num`	`int`	`1234`	`input_num >= 0`

Функция должна возвращать количество цифр в `input_num`.

Дополнительные условия:

- функция должна быть рекурсивной;
- функция не должна преобразовывать целое в строку.

Заготовка кода

```
def get_number_of_digits(input_num: int) -> int:
    # Здесь должна быть ваша реализация

print(get_number_of_digits(1234))
print(get_number_of_digits(0))
print(get_number_of_digits(123456789))
```

Примеры

```
Вход:  1234
Выход: 4

Вход: 0
Выход: 1

Вход: 123456789
Выход: 9
```

Задача 19

Предварительные сведения

В игру Tic-Tac-Toe (по-другому, крестики-нолики) два игрока играют на поле размером 3×3. Цель игры – первым поставить свои значки в ряд: по горизонтали, по вертикали или по диагонали. Игроки ходят по очереди, и каждый ставит свой значок (X или O) в пустую клетку, пока один из них не выиграет или все поле не окажется заполненным, что означает ничью.

Поле для игры в крестики-нолики можно представить двумерным списком Python, в котором внутренние списки представляют строки. Рассмотрим пример. Список

```
input_board = [["X", "X", "X"],
               ["0", "X", "0"],
               ["X", "0", "0"]
```

представляет следующую игровую позицию:

```
X X X
0 X 0
X 0 0
```

Задание

Определите функцию *get_tic_tac_toe_winner*, принимающую один параметр:

Имя	Тип	Пример входа
`input_board`	список списков строк	`[["X", "X", "X"], ["O", "X", "O"], ["X", "O", "O"]]`

Функция должна определять, кто выиграл. В случае ничьей она должна вернуть `None`.

Заготовка кода

```
def get_tic_tac_toe_winner(input_board: list[list[str]]) -> str | None:
    # Здесь должна быть ваша реализация

print(get_tic_tac_toe_winner([["X", "X", "X"], ["O", "X", "O"], ["X", "O", "O"]]))
print(get_tic_tac_toe_winner([["X", "O", "O"], ["O", "O", ""], ["X", "O", "O"]]))
print(get_tic_tac_toe_winner([["X", "O", "O"], ["O", "X", ""], ["X", "O", "O"]]))
```

Примеры

```
Вход:  [["X", "X", "X"], ["O", "X", "O"], ["X", "O", "O"]]
Выход: "X"

Вход: [["X", "O", "O"], ["O", "O", ""], ["X", "O", "O"]]
Выход: "O"

Вход: [["X", "O", "O"], ["O", "X", ""], ["X", "O", "O"]]
Выход: None
```

Задача 20

Задание

Определите функцию *print_triangle*, принимающую два параметра:

Имя	Тип	Пример входа
number_of_levels	int	4
symbol	str	"*"

Функция должна вывести симметричный относительно вертикальной оси треугольник, составленный из указанных символов. Количество символов в каждой строке должно увеличиваться на два при переходе к следующему уровню: на первом уровне должен быть один символ, на втором – три и т. д., пока не будет выведен конечный уровень. Например, при `number_of_levels = 4` и `symbol = "*"` должен получиться такой треугольник:

```
   *
  ***
 *****
*******
```

Заготовка кода

```
def print_triangle(number_of_levels: int, symbol: str) -> None:
    # Здесь должна быть ваша реализация

print_triangle(3, "*")
print_triangle(1, "|")
```

Примеры

```
Входы:
 - number_of_levels = 3
 - symbol = "*"

Выход:
   *
  ***
 *****

Входы:
 - number_of_levels = 1
 - symbol = "|"

Выход:
|
```

Задача 21

Предварительные сведения

Хорошо известная математическая последовательность чисел Фибоначчи много веков зачаровывала математиков и ученых. Она начинается числами 0, 1, а каждое следующее число равно сумме двух предыдущих.

Начало последовательности выглядит так: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

Задание

Определите функцию *fibonacci*, принимающую один параметр:

Имя	Тип	Пример входа
sequence_number	int	4

Функция должна вернуть число Фибоначчи с указанным порядковым номером (нумерация начинается с 0).

Дополнительные условия:

- функция должна быть рекурсивной;

Заготовка кода

```
def fibonacci(sequence_number: int) -> int:
    # Здесь должна быть ваша реализация

print(fibonacci(4))
print(fibonacci(0))
print(fibonacci(6))
```

Примеры

```
Вход: 4
Выход: 3

Вход: 0
Выход: 0

Вход: 6
Выход: 8
```

Задача 22

Предварительные сведения

Гармонической суммой называется сумма чисел, обратных заданным. Обратным числом называется результат деления 1 на это число. Например, обратным к 2 является 0.5, потому что 1, поделенная на 2, равна 0.5.

Обычно, говоря о гармонической сумме, имеют в виду множество натуральных чисел от 1 до n. Формула гармонической суммы n слагаемых имеет вид Hn = 1/1 + 1/2 + 1/3 + ... + 1/n.

Задание

Определите функцию *harmonic_sum*, принимающую один параметр:

Имя	Тип	Пример входа
N	int	5

Функция должна возвращать гармоническую сумму n слагаемых.

Дополнительные условия:

- функция должна быть рекурсивной;

Заготовка кода

```
def harmonic_sum(n: int) -> float:
    # Здесь должна быть ваша реализация

print(harmonic_sum(5))
print(harmonic_sum(2))
print(harmonic_sum(0))
```

Примеры

```
Вход: 5
Выход: 2.283

Вход: 2
Выход: 1.5

Вход: 0
Выход: 0
```

Задача 23

Предварительные сведения

Логический вентиль XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ) – это цифровая схема, которая выполняет логическую операцию над двумя входами. Выход вентиля XOR равен true (1) тогда и только тогда, когда один вход равен true, а другой false (0). Иными словами, вентиль XOR сравнивает входы, и если они различны, то формирует на выходе true. Если же входы одинаковы, то на выходе формируется false.

Это можно представить таблицей истинности, т. е. таблицей, в которой показаны выводы логического вентиля для всех возможных комбинаций входов. Вот пример таблицы истинности для вентиля XOR:

Вход A	Вход B	Выход
0	0	0
0	1	1
1	0	1
1	1	0

В этом примере левый столбец представляет вход A, средний – вход B, а правый – выход. Как видим, когда входы различны, выход равен 1, а когда совпадают – 0.

Задание

Определите функцию *xor*, принимающую два параметра:

Имя	Тип	Пример входа
`input_a`	`str`	`"1101"`
`input_b`	`str`	`"0001"`

Функция должна возвращать строку, являющуюся результатом применения XOR к двум входным строкам. Если одна строка длиннее другой, то лишние символы следует игнорировать.

Заготовка кода

```
def xor(input_a: str, input_b: str) -> str:
    # Здесь должна быть ваша реализация

print(xor("1111", "1111"))
```

```
print(xor("1111", "0000"))
print(xor("1101", "00010"))
```

Примеры

```
Входы:
 - input_a: "1111"
 - input_b: "1111"

Выход: "0000"

Входы:
 - input_a: "1111"
 - input_b: "0000"

Выход: "1111"

Входы:
 - input_a: "1101"
 - input_b: "00010"

Выход: "1100"
```

Задача 24

Предварительные сведения

В Python функция `zip` принимает один или несколько итерируемых объектов (например, списков, кортежей или строк) и возвращает итератор по кортежам. Каждый кортеж содержит элементы, взятые из каждого итерируемого объекта в одной и той же позиции. Функция `zip` останавливается по достижении конца самого короткого итерируемого объекта.

Ниже приведен пример работы функции `zip`:

```
a = [1, 2, 3]
b = [4, 5, 6]

zipped = zip(a, b)
print(list(zipped)) # [(1,4), (2, 5), (3, 6)]
```

Задание

Определите функцию *my_zip*, принимающую два параметра:

Имя	Тип	Пример входа
`input_list_a`	список `Any`	`[1, 2, 3, 4]`
`input_list_b`	список `Any`	`[5, 6, 7, 8]`

Функция должна возвращать такой же результат, как встроенная в Python функция `zip`.

Заготовка кода

```
from typing import Any

def my_zip(input_list_a: list[Any], input_list_b: list[Any])
    -> list[tuple[Any, Any]]:
    # Здесь должна быть ваша реализация

print(my_zip([1, 2, 3, 4], [5, 6, 7, 8]))
print(my_zip([], []))
print(my_zip([1, 2, 3], [5, 6, 7, 8]))
```

Примеры

```
Входы:
 - input_list_a: [1, 2, 3, 4]
 - input_list_b: [5, 6, 7, 8]

Выход: [(1, 5), (2, 6), (3, 7), (4, 8)]
```

```
Входы:
 - input_list_a: []
 - input_list_b: []

Выход: []

Входы:
 - input_list_a: [1, 2, 3]
 - input_list_b: [5, 6, 7, 8]

Выход: [(1, 5), (2, 6), (3, 7)]
```

Задача 25

Задание

Определите функцию *is_valid_equation*, принимающую один параметр:

Имя	Тип	Пример входа	Ограничение
`input_equation`	`str`	`"2 + 3 = 5"`	Строка должна состоять из целого числа, за которым следует знак плюс или минус, за ним еще одно целое число, знак равенства и ответ. Числа и знаки операций должны быть разделены пробелами

Функция должна возвращать булево значение, показывающее, верно ли равенство. Равенство считается верным, если

- его формат совпадает с описанным выше;
- обе части равенства равны одному и тому же числу.

Заготовка кода

```
def is_valid_equation(input_equation: str) -> bool:
    # Здесь должна быть ваша реализация

print(is_valid_equation("2 + 3 = 5"))
print(is_valid_equation("-5 + -6 = -11"))
print(is_valid_equation("-2 + 3 = -5"))
```

Примеры

```
Вход: "2 + 3 = 5"
Выход: True

Вход: "-5 + -6 = -11"
Выход: True

Вход: "-2 + 3 = -5"
Выход: False
```

Задача 26

Задание

Определите функцию *rotate_list_left*, принимающую два параметра:

Имя	Тип	Пример входа
`input_list`	список `Any`	`[1, 2, 3, 4, 5]`
`rotate_amount`	`int`	`2`

Функция должна возвращать новый список, элементами которого являются элементы исходного списка, циклически сдвинутые на заданное число позиций.

Дополнительные условия:

- в решении не должны использоваться циклы;
- функция должна работать, даже если величина сдвига больше длины списка, например результат циклического сдвига на 6 должен порождать такой же результат, как циклический сдвиг на 1.

Заготовка кода

```
from typing import Any

def rotate_list_left(input_list: list[Any], rotate_amount: int)
    -> list[Any]:
    # Здесь должна быть ваша реализация

print(rotate_list_left([1, 2, 3, 4, 5], 2))
print(rotate_list_left([1, 2, 3, 4, 5], 5))
print(rotate_list_left([1, 2, 3, 4, 5], 7))
```

Примеры

```
Входы:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 2
Выход: [3, 4, 5, 1, 2]

Входы:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 5
Выход: [1, 2, 3, 4, 5]

Входы:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 7
Выход: [3, 4, 5, 1, 2]
```

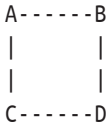
Задача 27

Предварительные сведения

Неориентированный граф – это математическое представление множества объектов, некоторые элементы которого связаны между собой. Объекты представляются вершинами (или узлами), а связи – ребрами. В неориентированном графе ребра не имеют направления, т. е. их можно проходить в обе стороны. Это означает, что если имеется ребро, соединяющее вершины A и B, то имеется также ребро, соединяющее вершины B и A.

Простым примером неориентированного графа является группа городов и соединяющих их дорог. Каждый город можно представить вершиной, а дороги между городами – ребрами. Не важно, движетесь вы из города A в город B или из города B в город A, – дорога между ними одна и та же.

Графически это можно изобразить так:



Здесь каждая буква представляет вершину (город) и линии – ребра (дороги). A и B соединены ребром, равно как A и C.

В Python графы обычно представляются матрицей смежности. Идея проста: мы располагаем список вершин сверху и такой список же список вершин сбоку. Если две вершины соединены ребром, то в клетку на пересечении соответствующих строки и столбца записывается 1, а если не соединены, то 0.

Пример: вершина A соединена с самой собой, вершиной B и вершиной C. Вершина B соединена только с вершиной A. Вершина C соединена только с вершиной A.

	Вершина A	**Вершина B**	**Вершина C**
Вершина A	1	1	1
Вершина B	1	0	0
Вершина C	1	0	0

В Python такую структуру можно представить списком списков, в котором каждой вершине сопоставлен индекс. Например, вершины A, B, C будут представлены индексами 0, 1, 2 соответственно:

```
graph = [
    [1, 1, 1],
    [1, 0, 0],
    [1, 0, 0]
]

graph[0][0]  == 1 # Вершина A соединена сама с собой?
graph[0][1]  == 1 # Вершина A соединена с вершиной B?
graph[2][1]  == 1 # Вершина C соединена с вершиной B?
```

Задание

Определите функцию *find_adjacent_nodes*, принимающую два параметра:

Имя	Тип	Пример входа
adj_matrix	список списков int	[[1, 1, 1], [1, 0, 0], [1, 0, 0]]
start_node	int	0

Функция должна возвращать список всех вершин, смежных со `start_node`.

Эту задачу можно решить функцией, тело которой содержит всего одну строку. Сможете ли вы найти такое решение?

Заготовка кода

```
def find_adjacent_nodes(
    adj_matrix: list[list[int]],
    start_node: int
) -> list[int]:

    # Здесь должна быть ваша реализация

print(find_adjacent_nodes([[1, 1, 1], [1, 0, 0], [1, 0, 0]], 0))
print(find_adjacent_nodes([[1, 1, 1], [1, 0, 0], [1, 0, 0]], 1))
print(find_adjacent_nodes([[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 1], [0, 1, 1,
0]], 1))
```

Примеры

```
Входы:
 - adj_matrix: [[1, 1, 1], [1, 0, 0], [1, 0, 0]]
 - start_node: 0
Выход: [0, 1, 2]

Входы:
 - adj_matrix: [[1, 1, 1], [1, 0, 0], [1, 0, 0]]
 - start_node: 1
Выход: [0]

Входы:
 - adj_matrix: [[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 1], [0, 1, 1, 0]]
 - start_node: 1
Выход: [0, 3]
```

Задача 28

Предварительные сведения

В техническом анализе пиком называется высшая точка, или локальный максимум, цены актива. Наоборот, впадиной называется низшая точка, или локальный минимум, цены актива. Когда трейдер смотрит на график изменения цены актива, пики и впадины помогают ему определить потенциальные поворотные точки и потенциальные изменения общего тренда актива.

Для наших целей определим пики и впадины следующим образом:

- пик:
  - к пику должна подходить слева возрастающая последовательность из одного или нескольких чисел;
  - от пика должна отходить справа убывающая последовательность из одного или нескольких чисел;
  - пик не может иметь место в начале или в конце движения цены;
- впадина:
  - к впадине должна подходить слева убывающая последовательность из одного или нескольких чисел;
  - от впадины должна отходить справа возрастающая последовательность из одного или нескольких чисел;
  - впадина не может иметь место в начале или в конце движения цены.

Задание

Определите функцию *count_peaks_valleys*, принимающую один параметр:

Имя	Тип	Пример входа
`price_action`	список `int`	`[1, 2, 3, 2, 1]`

Функция должна возвращать кортеж, показывающий, сколько имеется пиков и впадин в заданном движении цены. Кортеж должен содержать два целых числа, первое из которых равно числу пиков, а второе – числу впадин.

Заготовка кода

```
def count_peaks_valleys(price_action: list[int]) -> tuple[int, int]:
    # Здесь должна быть ваша реализация

print(count_peaks_valleys([1, 2, 3, 2, 1]))
print(count_peaks_valleys([1, 2, 3, 2, 1, 2]))
print(count_peaks_valleys([7, 6, 5, 10, 11, 12, 10, 9, 10]))
```

Примеры

```
Вход: [1, 2, 3, 2, 1]
Выход: (1, 0)

Вход: [1, 2, 3, 2, 1, 2]
Выход: (1, 1)

Вход: [7, 6, 5, 10, 11, 12, 10, 9, 10]
Выход: (1, 2)
```

Задача 29

Предварительные сведения

Кодом для перестукивания называется простой метод передачи сообщений с помощью серии постукиваний. Первоначально он был разработан военнопленными для секретного общения друг с другом, но использовался также в других ситуациях, когда связь затруднена, например заблудившимися туристами или людьми, погребенными под рухнувшим зданием. В коде для перестукивания используется следующая таблица 5×5 для представления каждой буквы:

	1	**2**	**3**	**4**	**5**
1	A	B	C/K	D	E
2	F	G	H	I	J
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Примечание: буквы C и K представлены одной ячейкой (1, 3) – при декодировании можете выбрать любую букву.

Для передачи сообщения отправитель выстукивает номера строки и столбца каждой буквы сообщения, делая паузы между буквами и более длинные паузы между словами. Получатель использует описанный код для декодирования сообщения.

Например, слово «water» представляется последовательностью:

- 5, 2
- 1, 1
- 4, 4
- 1, 5
- 4, 2

Или же такой серией стуков: ••••• •• • • •••• •••• • ••••• •••• ••

Задание

Определите функцию *tap_code_to_english*, принимающую один параметр:

Имя	**Тип**	**Пример входа**
`input_code`	`str`	`".. ... .. .... .. ... .. ...."`

Функция должна возвращать закодированное предложение. Элементы внутри строки должны быть представлены следующим образом:

- буквы: 1–5 точек, за которыми следует пробел, и далее еще 1–5 точек, например «. ..» = «b»;
- конец буквы и начало новой буквы: два пробела, например «.. ...» = «bb»;
- конец слова и начало нового слова: три пробела, например «.. ... .» = «hi hi».

Заготовка кода

```
tap_code_map = {
    "a": ". .",
    "b": ". ..",
    "c": ". ...",
    "d": ". ....",
    "e": ". .....",
    "f": ".. .",
    "g": ".. ..",
    "h": ".. ...",
    "i": ".. ....",
    "j": ".. .....",
    "l": "... .",
    "m": "... ..",
    "n": "... ...",
    "o": "... ....",
    "p": "... .....",
    "q": ".... .",
    "r": ".... ..",
    "s": ".... ...",
    "t": ".... ....",
    "u": ".... .....",
    "v": "..... .",
    "w": "..... ..",
    "x": "..... ...",
    "y": "..... ....",
    "z": "..... ....."
}

def tap_code_to_english(input_code: str) -> str:
    # Здесь должна быть ваша реализация

print(tap_code_to_english(".. ...  .. ....   .. ...  .. ...."))
print(tap_code_to_english(". ...  ... ....  ... ....  ... ."))
print(tap_code_to_english(""))
```

Примеры

```
Вход: ".. ...  .. ....   .. ...  .. ...."
Выход: "hi hi"

Вход: ". ...  ... ....  ... ....  ... ."
Выход: "cool"

Вход: ""
Выход: ""
```

Задача 30

Задание

Определите функцию *find_zero_sum_triplets*, принимающую один параметр:

Имя	Тип	Пример входа
`input_nums`	список `int`	`[1, 2, 3, 4, 5, -9]`

Функция должна возвращать все возможные комбинации индексов трех чисел, которые в сумме дают 0. Каждая комбинация должна быть представлена кортежем, содержащим три индекса, а функция должна возвращать список таких кортежей или пустой список, если ни одной нужной комбинации не существует. Функция должна корректно обрабатывать повторяющиеся числа во входном списке.

Заготовка кода

```
def find_zero_sum_triplets(input_nums: list[int])
    -> list[tuple[int, int, int]]:
    # Здесь должна быть ваша реализация

print(find_zero_sum_triplets([1, 2, 3, 4, 5]))
print(find_zero_sum_triplets([1, 2, 3, 4, 5, -9]))
print(find_zero_sum_triplets([1, 2, 3, 4, 5, -9, -9]))
```

Примеры

```
Вход: [1, 2, 3, 4, 5]
Выход: []

Вход: [1, 2, 3, 4, 5, -9]
Выход: [(3, 4, 5)]

Вход: [1, 2, 3, 4, 5, -9, -9]
Выход: [(3, 4, 5), (3, 4, 6)]
```

Задача 31

Задание

Определите функцию *param_count*, принимающую произвольное число параметров.

Функция должна возвращать число переданных ей аргументов.

Заготовка кода

```
from typing import Any

def param_count(*args: Any) -> int:
    # Здесь должна быть ваша реализация

print(param_count(1, 2, 3, 4, 5))
print(param_count("hello"))
print(param_count())
```

Примеры

```
Входы: 1, 2, 3, 4, 5
Выход: 5

Входы: "hello"
Выход: 1

Вход:
Выход: 0
```

Задача 31.1 (дополнительная)

Задание

Улучшите функцию *my_zip* из задачи 24, так чтобы она могла принимать произвольное число аргументов.

Функция должна возвращать тот же результат, что и встроенная функция *zip*. Ваша функция может возвращать список, или если вы хотите максимально точно следовать встроенной в Python функции, то она может возвращать итератор.

Заготовка кода

Вариант с возвратом списка.

```
from typing import Any

def my_zip_one(*input_lists: list[Any]) -> list[tuple[Any, ...]]:
    # Здесь должна быть ваша реализация

print(my_zip_one([1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]))
print(my_zip_one([1, 2, 3], [5, 6, 7, 8]))
print(my_zip_one([], []))
```

Вариант с возвратом итератора.

```
from typing import Any
from collections.abc import Iterator

def my_zip_two(*input_lists: list[Any])
    -> Iterator[tuple[Any, ...]]:
    # Your implementation here

print(list(my_zip_two([1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12])))
print(list(my_zip_two([1, 2, 3], [5, 6, 7, 8])))
print(list(my_zip_two([], [])))
```

Примеры

```
Входы: [1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]
Выход: [(1, 5, 9], [2, 6, 10), (3, 7, 11), (4, 8, 12)]

Входы: [1, 2, 3], [5, 6, 7, 8]
Выход: [(1, 5], [2, 6), (3, 7)]

Вход: [], []
Выход: []
```

Задача 32

Задание

Определите функцию *contains_python_chars*, принимающую один параметр:

Имя	Тип	Пример входа
`input_str`	`str`	`"Nohtyp"`

Функция должна возвращать булево значение, показывающее, содержит ли строка какую-нибудь анаграмму слова «python» без учета регистра.

Заготовка кода

```
def contains_python_chars(input_str: str) -> bool:
    # Здесь должна быть ваша реализация

print(contains_python_chars("pYThon"))
print(contains_python_chars("Nohtyp"))
print(contains_python_chars("pythZon"))
```

Примеры

```
Вход: "pYThon"
Выход: True

Вход: "Nohtyp"
Выход: True

Вход: "pythZon"
Выход: False
```

Задача 33

Предварительные сведения

Целое положительное число называется простым, если оно не имеет иных делителей, кроме 1 и самого себя. Вот несколько примеров простых чисел: 2, 3, 5, 7, 11, 13, 17, 19. Они играют важную роль в теории чисел и имеют много приложений в таких областях, как криптография, теория кодирования и построение генераторов псевдослучайных чисел.

Задание

Определите функцию *find_primes*, принимающую один параметр:

Имя	Тип	Пример входа
`input_nums`	список `int`	`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

Функция должна возвращать новый список, содержащий только простые числа в списке *input_nums*.

Заготовка кода

```
def find_primes(input_nums: list[int]) -> list[int]:
    # Здесь должна быть ваша реализация

print(find_primes([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
print(find_primes([-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]))
print(find_primes([2, 3, 5, 7, 11, 13, 17]))
```

Примеры

```
Вход: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Выход: [2, 3, 5, 7]

Вход: [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
Выход: []

Вход: [2, 3, 5, 7, 11, 13, 17]
Выход: [2, 3, 5, 7, 11, 13, 17]
```

Задача 34

Предварительные сведения

ROT13 – простой метод шифрования, при котором каждая буква сообщения заменяется буквой, отстоящей от нее на 13 позиций в алфавите. Например, буква «A» становится «N», «B» становится «O» и т. д.

Для применения шифра ROT13 к заданному сообщению нужно просто сдвинуть каждую букву на 13 позиций вперед. Если для полного сдвига не хватает букв, то нужно вернуться в начало алфавита и продолжить сдвиг оттуда.

Задание

Определите функцию *rot13*, принимающую один параметр:

Имя	Тип	Пример входа
`input_str`	`str`	`"Hello world!"`

Функция должна возвращать новую строку, зашифрованную шифром ROT13. Числа и другие символы, отличные от букв, при этом не должны изменяться.

Заготовка кода

```
def rot13(input_str: str) -> str:
    # Здесь должна быть ваша реализация

print(rot13("Hello world!"))
print(rot13("Cool puzzles!"))
print(rot13("12345!@£$%"))
```

Примеры

```
Вход: "Hello world!"
Выход: "Uryyb jbeeq!"

Вход: "Cool puzzles!"
Выход: "Pbby chmmyrf!"

Вход: "12345!@£$%"
Выход: "12345!@£$%"
```

Задача 35

Задание

Определите функцию *get_parentheses_groups*, принимающую один параметр:

Имя	Тип	Пример входа	Ограничение
`input_str`	`str`	`"(( )) (( ) ) ( (( )))"`	Скобки в строке должны быть парными, т. е. за каждыми N открывающими скобками должно следовать N закрывающих скобок

Функция должна возвращать список, содержащий полные группы парных скобок без пробелов. Каждой группе скобок должен соответствовать один элемент списка.

Заготовка кода

```
def get_parentheses_groups(input_str: str) -> list[str]:
    # Здесь должна быть ваша реализация

print(get_parentheses_groups("(( ))  ((  ) ) (   ((  )))"))
print(get_parentheses_groups("( ( ( ( ( ( ) ) ) ) ) )"))
print(get_parentheses_groups(""))
```

Примеры

```
Вход: "(( ))  ((  ) ) (   ((  )))"
Выход: ["(())", "(())", "((()))"]

Вход: "( ( ( ( ( ( ) ) ) ) ) )"
Выход: ["((((((()))))))"]

Вход: ""
Выход: []
```

Задача 36

Предварительные сведения

В математике матрицей называют прямоугольную таблицу чисел, символов или выражений, состоящую из строк и столбцов. Матрицы часто обозначают заглавными буквами, например A, B, C и т. д. Каждый элемент матрицы идентифицируется индексами строки и столбца.

Например, рассмотрим следующую матрицу A:

```
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]
```

Эта матрица состоит из 3 строк и 3 столбцов, а ее элементы можно обозначить `A_ij`, где `i` – индекс строки, а `j` – индекс столбца. Так что в нашем примере `A_12 = 2` и `A_31 = 7`.

Матрицы можно использовать для представления больших наборов данных, и над ними можно производить операции, например умножение.

Для умножения двух матриц необходимо, чтобы число столбцов левой матрицы было равно числу строк правой. Если это условие выполнено, то можно поступить следующим образом.

1. Определить размер матрицы-произведения: в ней будет столько строк, сколько в первой матрице, и столько столбцов, сколько во второй.
2. Создать матрицу такого размера и обнулить все ее элементы.
3. Для каждого элемента `C_ij` матрицы-произведения перемножить соответственные элементы `i`-й строки первой матрицы и `j`-го столбца второй матрицы.
4. Просуммировать произведения, вычисленные на предыдущем шаге, и присвоить результат соответствующему элементу матрицы-произведения.
5. Повторить шаги 3 и 4 для каждого элемента матрицы-произведения.
6. Вернуть результирующую матрицу.

Пример:

```
A = [[2, 3], [4, 5]]
B = [[10, 15], [5, 1]]
C = [[0, 0], [0, 0]]
```

```
C[0][0] = (A[0][0] * B[0][0]) + (A[0][1] * B[1][0])
C[0][1] = (A[0][0] * B[0][1]) + (A[0][1] * B[1][1])
C[1][0] = (A[1][0] * B[0][0]) + (A[1][1] * B[1][0])
C[1][1] = (A[1][0] * B[0][1]) + (A[1][1] * B[1][1])
```

```
Выход: [[35, 33], [65, 65]]
```

Задание

Определите функцию *matrix_multiply*, принимающую два параметра:

Имя	Тип	Пример входа
`left_matrix`	список списков `int`	`[[2, 3], [4, 5]]`
`right_matrix`	список списков `int`	`[[10, 15], [5, 1]]`

Функция должна возвращать результат умножения левой и правой матриц.

Заготовка кода

```
A = [[2, 3], [4, 5]]
B = [[10, 15], [5, 1]]

def matrix_multiply(
    left_matrix: list[list[int]], right_matrix: list[list[int]]
) -> list[list[int]]:

    # Здесь должна быть ваша реализация

print(matrix_multiply(A, B))
print(matrix_multiply([[1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6]],
                      [[1, 2, [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]]))
print(matrix_multiply([[1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6]], [[1, 2, 3]]))
```

Примеры

```
Входы:
 - left_matrix: [[2, 3], [4, 5]]
 - right_matrix: [[10, 15], [5, 1]]

Выход: [[35, 33], [65, 65]

Входы:
 - left_matrix: [1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6]
 - right_matrix: [[1, 2, [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]]

Выход: [[161, 182], [161, 182]

Входы:
 - left_matrix: [1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6]
 - right_matrix: [[1, 2, 3]]

Выход: None
```

Задача 37

Предварительные сведения

Наибольшим общим делителем двух или более целых чисел называется наибольшее положительное число, на которое все числа делятся без остатка. Например, НОД 8 и 12 равен 4, потому что 4 – наибольшее число, на которое делятся 8 и 12.

Задание

Определите функцию *gcd*, принимающую два параметра:

Имя	Тип	Пример входа
num_one	Int	36
num_two	Int	8

Функция должна возвращать наибольший общий делитель двух целых чисел.

Заготовка кода

```
def gcd(num_one: int, num_two: int) -> int:
    # Здесь должна быть ваша реализация

print(gcd(36, 8))
print(gcd(5, 25))
print(gcd(5, 26))
```

Примеры

```
Входы:
 - num_one: 36
 - num_two: 8
Выход: 4

Входы:
 - num_one: 5
 - num_two: 25
Выход: 5

Входы:
 - num_one: 5
 - num_two: 26
Выход: 1
```

Задача 38

Задание

Определите функцию *find_pairs_summing_to_target*, принимающую два параметра:

Имя	Тип	Пример входа
`input_nums`	список `int`	`[1, 2, 3, 4, 5, 6, 7, 8, 9]`
`target`	`int`	`int`

Функция должна возвращать список пар в `input_nums`, сумма которых равна `target`.

Дополнительное условие:

- результирующий список не должен содержать дубликатов, т. е. если `input_nums=[5, 5, 5, 5]` и `target=10`, то результат должен быть равен `[(5,5)]`, а не `[(5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5)]`.

Заготовка кода

```
def find_pairs_summing_to_target(
    input_nums: list[int], target: int
) -> list[tuple[int, int]]:
    # Здесь должна быть ваша реализация

print(find_pairs_summing_to_target([5, 5, 5, 5], 10))
print(find_pairs_summing_to_target([1, 2, 3, 4, 5, 6, 7, 8, 9], 10))
print(find_pairs_summing_to_target([11, 12, 13, 14, 15], 5))
```

Примеры

```
Входы:
 - input_nums: [5, 5, 5, 5]
 - target: 10
Выход: [(5, 5)]

Входы:
 - input_nums: [1, 2, 3, 4, 5, 6, 7, 8, 9]
 - target: 10
Выход: [(1, 9)], (2, 8), (3, 7), (4, 6)]

Входы:
 - input_nums: [11, 12, 13, 14, 15]
 - target: 5
Выход: []
```

Задача 38.1 (дополнительная)

Задание

Улучшите функцию *find_pairs_summing_to_target* из задачи 38, так чтобы в ней не было вложенных циклов. Входы и выходы остаются прежними. Если в вашем решении и так нет вложенных циклов, то пропустите эту задачу.

Заготовка кода

```
def find_pairs_summing_to_target_bonus(
    input_nums: list[int], target: int
) -> list[tuple[int, int]]:

    # Здесь должна быть ваша реализация

print(find_pairs_summing_to_target_bonus([5, 5, 5, 5], 10))
print(find_pairs_summing_to_target_bonus([1, 2, 3, 4, 5, 6, 7, 8, 9], 10))
print(find_pairs_summing_to_target_bonus([11, 12, 13, 14, 15], 5))
```

Примеры

```
Входы:
 - input_nums: [5, 5, 5, 5]
 - target: 10
Выход: [(5, 5)]

Входы:
 - input_nums: [1, 2, 3, 4, 5, 6, 7, 8, 9]
 - target: 10
Выход: [(1, 9)], (2, 8), (3, 7), (4, 6)]

Входы:
 - input_nums: [11, 12, 13, 14, 15]
 - target: 5
Выход: []
```

Задача 39

Предварительные сведения

В задаче о ханойской башне имеется три стержня и сколько-то дисков разных размеров, которые можно насаживать на любой стержень. В начале диски насажены в порядке возрастания размера на один диск, так что самый маленький диск находится сверху, т. е. стопка имеет коническую форму.

Требуется переместить всю стопку дисков на другой стержень, соблюдая следующие правила:

1. За один ход можно перемещать только один диск.
2. На каждом ходе с одного стержня снимается верхний диск и кладется поверх стопки на другом стержне или на пустой стержень.
3. Нельзя класть диск на диск меньшего размера.

Задание

Напишите функцию *tower_of_hanoi*, принимающую четыре параметра.

Имя	Тип	Пример входа	Описание
`num_disks`	`int`	`4`	Число дисков на исходном стержне в начале задачи
`source`	`str`	`"Source"`	Имя исходного стержня
`aux`	`str`	`"Auxiliary"`	Имя вспомогательного стержня
`target`	`str`	`"Target"`	Имя конечного стержня

Функция должна решать задачу о ханойской башне. Результат должен быть напечатан на консоли в следующем формате: «Переместить диск {n} с {стержень} на {другой_стержень}», где n – размер диска. Наименьший диск имеет размер 1, следующий за ним – размер 2 и т. д.

Заготовка кода

```
def tower_of_hanoi(num_disks: int, source: str, aux: str, target: str) -> None:
    # Здесь должна быть ваша реализация

tower_of_hanoi(2, "Source", "Auxiliary", "Target")
print("---")
tower_of_hanoi(4, "Source", "Auxiliary", "Target")
```

Примеры

```
Входы:
 - num_disks: 2
 - source: "Source"
 - aux: "Auxiliary"
 - target: "Target"
Выход:
Переместить диск 1 с Source на Auxiliary
Переместить диск 2 с Source на Target
Переместить диск 1 с Auxiliary на Target
```

```
Входы:
 - num_disks: 4
 - source: "Source"
 - aux: "Auxiliary"
 - target: "Target"
Выход:
Переместить диск 1 с Source на Auxiliary
Переместить диск 2 с Source на Target
Переместить диск 1 с Auxiliary на Target
Переместить диск 3 с Source на Auxiliary
Переместить диск 1 с Target на Source
Переместить диск 2 с Target на Auxiliary
Переместить диск 1 с Source на Auxiliary
Переместить диск 4 с Source на Target
Переместить диск 2 с Auxiliary на Source
Переместить диск 1 с Target на Source
Переместить диск 3 с Auxiliary на Target
Переместить диск 1 с Source на Auxiliary
Переместить диск 2 с Source на Target
Переместить диск 1 с Auxiliary на Target
```

Задача 40

Предварительные сведения

Сортировка вставками – простой алгоритм сортировки, который строит конечный отсортированный список, добавляя в него по одному элементу. Он обходит входной список и для каждого элемента сравнивает его с предшествующими элементами, а затем вставляет его в нужную позицию. Этот процесс продолжается, пока все элементы не будут сравнены и помещены в правильные позиции конечного отсортированного списка.

Название «сортировка вставками» объясняется тем, что алгоритм можно интерпретировать как вставку каждого элемента входного списка в правильную позицию выходного списка.

Примером сортировки вставки может служить сортировка колоды карт: сначала левая рука пуста, и мы удаляем по одной карте из колоды в правой руке, помещая ее в нужное место в левой руке.

Задание

Определите функцию *insertion_sort*, принимающую один параметр.

Имя	Тип	Пример входа
`input_nums`	список `int`	`[5, 10, 9, 11, 4]`

Функция должна реализовывать алгоритм сортировки вставками и возвращать отсортированный список `input_nums`.

Заготовка кода

```
def insertion_sort(input_nums: list[int]) -> list[int]:
    # Здесь должна быть ваша реализация

print(insertion_sort([5, 10, 9, 11, 4]))
print(insertion_sort([1, 2, 3, 4, 5]))
print(insertion_sort([-1, -2, -3, -4, -5]))
```

Примеры

```
Вход: [5, 10, 9, 11, 4]
Выход: [4, 5, 9, 10, 11]

Вход: [1, 2, 3, 4, 5]
Выход: [1, 2, 3, 4, 5]

Вход: [-1, -2, -3, -4, -5]
Выход: [-5, -4, -3, -2, -1]
```

Задача 41

Предварительные сведения

Римские числа – система счисления, возникшая в Римской империи. Это способ представления чисел с помощью букв латинского алфавита. В системе используется семь букв, каждой из которых сопоставлено числовое значение: I (1), V (5), X (10), L (50), C (100), D (500) и M (1000).

Для записи целого числа в римской системе счисления нужно выполнить следующие шаги.

1. Создать справочную таблицу для римских цифр (см. заготовку кода).
2. Найти наибольшую римскую цифру, меньшую или равную заданному целому числу.
3. Дописать эту римскую цифру в конец результирующей строки.
4. Вычесть соответствующее целое значение из исходного целого числа.
5. Повторять шаги 2, 3 и 4 для нового целого числа, пока не получится 0.
6. Вернуть результирующую строку.

Задание

Определите функцию *int_to_roman*, принимающую один параметр.

Имя	Тип	Пример входа	Ограничение
`input_num`	`int`	`3000`	Целое число должно принадлежать диапазону от 1 до 4999 включительно

Функция должна возвращать входное число, записанное в римской системе счисления. Помните о следующих ограничениях римской системы.

1. Разрешено использовать только римские цифры I, V, X, L, C, D и M и больше никаких других символов.
2. Необходимо придерживаться традиционного правила: помещение меньшей цифры перед большей означает вычитание. То есть 4 представляется как IV, а не как IIII.

Заготовка кода

```
roman_map = {
    1000: "M", 900: "CM", 500: "D", 400: "CD", 100: "C",
    90: "XC", 50: "L", 40: "XL", 10: "X", 9: "IX", 5: "V",
    4: "IV", 1: "I"
}

def int_to_roman(input_num: int) -> str:
    # Здесь должна быть ваша реализация

print(int_to_roman(4))
print(int_to_roman(27))
print(int_to_roman(4999))
```

Примеры

```
Вход: 4
Выход: "IV"

Вход: 27
Выход: "XXVII"

Вход: 4999
Выход: "MMMMCMXCIX"
```

Задача 41.1 (дополнительная)

Задание

Определите две дополнительные функции: *roman_to_int* и *int_roman_converter*.

Функция *roman_to_int* является обратной к определенной в задаче 41 функции *int_to_roman*. Она принимает строку, содержащую римское число, и возвращает соответствующее целое число.

Имя	Тип	Пример входа	Ограничение
`input_str`	`Str`	`"XXVII"`	Целое число должно принадлежать диапазону от 1 до 4999 включительно

Функция *int_roman_converter* принимает целое число или строку, содержащую число в римской записи. В зависимости от входа функция делает одно из двух:

- возвращает входное целое, записанное в римской нотации;
- возвращает входную строку, записанную в виде целого числа.

Чтобы определить, что именно ей передано на вход – целое число или строка, функция может воспользоваться встроенной в Python функцией `isinstance`.

Имя	Тип	Пример входа	Ограничение
`to_convert`	`str` или `int`	`"XXVII"`	Целое число должно принадлежать диапазону от 1 до 4999 включительно

Заготовка кода

```
# ... ваше решение задачи 41

def roman_to_int(input_str: str) -> int:
    # Здесь должна быть ваша реализация

def int_roman_converter(to_convert: str | int) -> int | str:
    # Здесь должна быть ваша реализация

for i in range(1, 5000):
    is_equal = int_roman_converter(int_roman_converter(i)) == i

    if not is_equal:
        print(f"{i} is incorrect!")
```

Примеры

```
# примеры для roman_to_int
Вход: "IV"
Выход: 4

Вход: "XXVII"
Выход: 27

Вход: "MMMMCMXCIX"
Выход: 4999

# примеры для int_roman_converter
Вход: "IV"
Выход: 4

Вход: 27
Выход: "XXVII"

Вход: "MMMMCMXCIX"
Выход: 4999
```

Задача 42

Задание

Определите функцию *bitwise_add*, принимающую два параметра.

Имя	Тип	Пример входа
num_one	int	3
num_two	int	4

Функция должна возвращать сумму двух входных параметров.

Дополнительное условие:

- функция не должна использовать никаких арифметических операторов.

Примечание: если вы незнакомы с двоичными числами, то прежде чем пытаться решить эту задачу, должны освоить основы. Поищите в Google (или в Яндекс) что-то вроде «introduction to binary numbers» («введение в двоичные числа»). Также почитайте о логических вентилях.

Заготовка кода

```
def bitwise_add(num_one: int, num_two: int) -> int:
    # Здесь должна быть ваша реализация

print(bitwise_add(3, 4))
print(bitwise_add(255, 256))
print(bitwise_add(-1, -2))
```

Примеры

```
Входы:
 - num_one: 3
 - num_two: 4
Выход: 7

Входы:
 - num_one: 255
 - num_two: 256
Выход: 511

Входы:
 - num_one: -1
 - num_two: -2
Выход: -3
```

Задача 43

Предварительные сведения

Двоичный поиск – это эффективный алгоритм нахождения элемента в отсортированном списке. Основная идея алгоритма – повторное деление интервала поиска пополам до тех пор, пока значение не будет найдено или интервал поиска не окажется пустым.

Двоичный поиск состоит из следующих шагов.

1. Установить интервал поиска совпадающим со всем списком.
2. Вычислить средний элемент. Если длина списка четна, то берем средний элемент с недостатком; например для списка [1, 2, 3, 4] средним будет элемент 2.
3. Сравнить средний элемент интервала поиска с искомым значением.
   a. Если средний элемент равен искомому значению, то алгоритм останавливается и возвращается индекс элемента.
   b. Если средний элемент больше искомого значения, то искомый элемент должен находиться в левой половине списка. Поэтому мы устанавливаем в качестве интервала поиска левую половину текущего интервала.
   c. Если средний элемент меньше искомого значения, то искомый элемент должен находиться в правой половине списка. Поэтому мы устанавливаем в качестве интервала поиска правую половину текущего интервала.
4. Повторять шаг 3 алгоритма с новым интервалом поиска, пока значение не будет найдено или интервал поиска не окажется пустым.

Проиллюстрируем эти шаги на простом примере.

```
Отсортированный список = [2, 3, 4, 10, 40]
Искомое значение = 10

# Шаг 1
Интервал поиска = отсортированный список
Средний элемент = 4

# Шаг 2
10 равно 1? Нет.
10 больше 4? Да.
10 меньше 4? Нет.
```

```
Интервал поиска = [10, 40]
Средний элемент = 10

10 равно 10? Да - значение найдено.
```

Задание

Определите функцию *binary_search*, принимающую два параметра.

Имя	Тип	Пример входа
sorted_list	Список int	[2, 3, 4, 10, 40]
value_to_find	int	10

Функция должна выполнить двоичный поиск в списке sorted_list и вернуть индекс значения value_to_find, если оно найдено. В противном случае функция должна вернуть –1.

Дополнительное условие:

- временная сложность функции должна быть равна *O(log n)*.

Заготовка кода

```
def binary_search(sorted_list: list[int], value_to_find: int) -> int:
    # Здесь должна быть ваша реализация

searchable_list = [2, 3, 4, 10, 40]

print(binary_search(searchable_list, 10))
print(binary_search(searchable_list, 0))
print(binary_search([], 0))
```

Примеры

```
Входы:
 - sorted_list: [2, 3, 4, 10, 40]
 - value_to_find: 10
Выход: 3

Входы:
 - sorted_list: [2, 3, 4, 10, 40]
 - value_to_find: 0
Выход: -1

Входы:
 - sorted_list: []
 - value_to_find: 0
Выход: -1
```

Задача 44

Предварительные сведения

Быстрая сортировка (Quicksort) – алгоритм типа «разделяй и властвуй», который используется для сортировки списков. Его идея заключается в том, чтобы выбрать «опорный» элемент списка, а остальные разделить на два подсписка, в один из которых входят элементы, большие опорного, а в другой – меньшие. Затем эти подсписки сортируются рекурсивно, и в конечном итоге весь список оказывается отсортированным.

Алгоритм состоит из следующих шагов.

1. Выбрать опорный элемент из списка. Этот элемент используется для разбиения списка на два подсписка.
2. Разбить список на две части, переместив все элементы, меньшие опорного, влево от него, а все элементы, большие опорного, вправо.
3. Рекурсивно отсортировать левый подсписок.
4. Рекурсивно отсортировать правый подсписок.
5. Объединив подсписки и опорный элемент, получить отсортированный список.

Эффективность алгоритма зависит от выбора опорного элемента. Если он выбран удачно, то алгоритм будет работать быстро, а если нет, то медленно. Обычно в качестве опорного выбирают первый, средний или последний элемент списка.

Временная сложность быстрой сортировки в среднем равна $O(n \log n)$, что делает ее весьма эффективным алгоритмом, в особенности для больших списков. А поскольку это алгоритм сортировки «на месте», он не требует дополнительной памяти, т. к. производит сортировку, меняя местами элементы исходного списка.

Задание

Определите функцию *quicksort*, принимающую три параметра.

Имя	Тип	Пример входа
`input_list`	Список `int`	`[5, 7, 8, 1, 2, 4, 99, 77, 56, 43, 12, 98]`
`low`	`int`	`0`
`high`	`int`	`11`

Функция должна сортировать `input_list`, применяя алгоритм быстрой сортировки.

Заготовка кода

```
def quicksort(input_list: list[int], low: int, high: int)
    -> list[int]:
    # Здесь должна быть ваша реализация

unsorted_list = [5, 7, 8, 1, 2, 4, 99, 77, 56, 43, 12, 98]
print(quicksort(unsorted_list, 0, len(unsorted_list) - 1))

unsorted_list = [10, 5, -10, -5, 0]
print(quicksort(unsorted_list, 0, len(unsorted_list) - 1))

print(quicksort([], 0, 0))
```

Примеры

```
Входы:
 - input_list: [5, 7, 8, 1, 2, 4, 99, 77, 56, 43, 12, 98]
 - low: 0
 - high: len(input_list) - 1
Выход: [1, 2, 4, 5, 7, 8, 12, 43, 56, 77, 98, 99]

Входы:
 - input_list: [10, 5, -10, -5, 0]
 - low: 0
 - high: len(input_list) - 1
Выход: [-10,- 5, 0, 5, 10]

Входы:
 - input_list: []
 - low: 0
 - high: 0
Выход: []
```

Задача 45

Предварительные сведения

Имеется набор предметов, каждый из которых характеризуется весом и ценностью, а также рюкзак ограниченной вместимости. Требуется найти комбинацию предметов, которая имела бы максимальную ценность, не превысив при этом вместимость рюкзака.

Предметы представлены списком кортежей, в каждом кортеже хранятся вес и ценность предмета. Вместимость рюкзака является входным параметром программы. Все предметы имеются в единственном экземпляре.

Задание

Определите функцию *solve_knapsack_problem*, принимающую два параметра.

Имя	Тип	Пример входа	Описание
`items`	Список кортежей	`[(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]`	`[(вес,ценность),...]`
`knapsack_capacity`	`int`	`5`	

Функция должна вернуть максимальную ценность, при условии что вместимость рюкзака не превышена.

Заготовка кода

```
def solve_knapsack_problem(
    items: list[tuple[int, int]], knapsack_capacity: int
) -> int:
    # Здесь должна быть ваша реализация

items = [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
max_weight = 5
print(solve_knapsack_problem(items, max_weight))

items = [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
max_weight = 0
print(solve_knapsack_problem(items, max_weight))

items = []
max_weight = 5
print(solve_knapsack_problem(items, max_weight))
```

Примеры

```
Входы:
 - items: [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
 - knapsack_capacity: 5
Выход: 2000
```

```
Входы:
 - items: [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
 - knapsack_capacity: 0
Выход: 0
```

```
Входы:
 - items: []
 - knapsack_capacity: 5
Выход: 0
```

Задача 45.1 (дополнительная)

Задание

Улучшите свое решение задачи об упаковке рюкзака 45, так чтобы его временная сложность составляла $O(nW)$, где n – количество предметов, а W – вместимость рюкзака.

Заготовка кода

```
def solve_knapsack_problem(
    items: list[tuple[int, int]], knapsack_capacity: int
) -> int:
    # Здесь должна быть ваша реализация

items = [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
max_weight = 5
print(solve_knapsack_problem(items, max_weight))

items = [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
max_weight = 0
print(solve_knapsack_problem(items, max_weight))

items = []
max_weight = 5
print(solve_knapsack_problem(items, max_weight))
```

Примеры

```
Входы:
 - items: [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
 - knapsack_capacity: 5
Выход: 2000

Входы:
 - items: [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
 - knapsack_capacity: 0
Выход: 0

Входы:
 - items: []
 - knapsack_capacity: 5
Выход: 0
```

Задача 46

Задание

Определите функцию *ip_range_to_list*, принимающую один параметр.

Имя	Тип	Пример входа	Описание
`input_ip_range`	`str`	`"192.255.255.0-192.255.255.255"`	Строка должна быть записана в формате: начальный IP-адрес, знак минуса, конечный IP-адрес

Функция должна возвращать список всех IP-адресов, принадлежащих диапазону. Каждый адрес должен быть представлен в виде строки в формате «x.x.x.x». IP-адреса должны следовать в числовом порядке.

Для максимальной эффективности в решении должны использоваться поразрядные операции. Воспользуйтесь двумя библиотеками: `struct` и `socket`.

Заготовка кода

```
import socket
import struct

def ip_range_to_list(input_ip_range: str) -> list[str]:
    # Здесь должна быть ваша реализация

print(ip_range_to_list("192.168.1.1-192.168.1.5"))
print(ip_range_to_list("1.1.1.0-1.1.1.1"))
print(ip_range_to_list("192.255.255.0-192.255.255.0"))
```

Примеры

```
Вход: "192.168.1.1-192.168.1.5"
Выход: ["192.168.1.1",
        "192.168.1.2",
        "192.168.1.3",
        "192.168.1.4",
        "192.168.1.5"]

Вход: "1.1.1.0-1.1.1.1"
Выход: ["1.1.1.0", "1.1.1.1"]

Вход: "192.255.255.0-192.255.255.0"
Выход: ["192.255.255.0"]
```

Задача 47

Задание

Определите функцию *solve_maze*, принимающую три параметра.

Имя	Тип	Пример входа	Описание
`maze`	Список списков `int`	`[[0, 1, 1], [0, 0, 1], [1, 0, 1]]`	Двумерный список, описывающий стены (1) и проходы (0). Каждый внутренний список описывает одну строку лабиринта, *n*-й элемент которой находится в *n*-м столбце
`start_pos`	Кортеж (`int`, `int`)	`(0,0)`	Координаты точки входа в лабиринт в формате (x, y)
`end_pos`	Кортеж (`int`, `int`)	`(1,2)`	Координаты точки выхода из лабиринта в формате (x, y)

Функция должна возвращать булево значение, показывающее, можно ли пройти лабиринт. Лабиринт является проходимым, если существует путь (состоящий только из нулей) из `start_pos` в `end_pos`. Разрешены только перемещения вверх, вниз, влево и вправо, перемещения по диагонали запрещены.

Заготовка кода

```
def solve_maze(
    maze: list[list[int]],
    start_pos: tuple[int, int],
    end_pos: tuple[int, int]
) -> bool:
    # Здесь должна быть ваша реализация

start = (0, 0)
end = (1, 2)

solvable_maze = [
    [0, 1, 1],
    [0, 0, 1],
    [1, 0, 1]
]

print(solve_maze(solvable_maze, start, end))
```

Примеры

```
# --- пример проходимого лабиринта ---#
maze = [
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 1, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
    [1, 1, 0, 0, 1, 0, 1, 1, 0, 1],
    [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
]
start = (1, 0)
end = (1, 9)

print(solve_maze(maze, start, end)) # возвращает True

# --- пример непроходимого лабиринта --- #
maze = [
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 1, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
    [1, 1, 0, 1, 1, 0, 1, 1, 0, 1],
    [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
]
start = (1, 0)
end = (1, 9)

print(solve_maze(maze, start, end)) # возвращает False
```

Задача 48

Предварительные сведения

В информатике двоичным деревом называется дерево, в котором каждый узел имеет нуль, один или два дочерних узла. Если узел имеет дочерние узлы, то они называются левым и правым потомками. Типичное действие – обход двоичного дерева, когда мы посещаем все узлы в определенном порядке и выводим хранящиеся в них значения.

Существует три основных метода обхода дерева: в прямом порядке, в симметричном порядке и в обратном порядке. В этой задаче мы рассмотрим симметричный порядок, который работает следующим образом.

1. Рекурсивно обойти левое поддерево, вызвав функцию обхода в симметричном порядке для левого потомка.
2. Посетить корневой узел.
3. Рекурсивно обойти правое поддерево, вызвав функцию обхода в симметричном порядке для правого потомка.

Задание

Определите функцию *traverse_inorder*, принимающую один параметр.

Имя	Тип	Пример входа
`root_node`	`TreeNode`	`TreeNode(4, two, six)`

`TreeNode` – созданный нами простой класс с тремя свойствами:

Имя	Тип	Пример входа
`val`	`int`	`1`
`left_node`	`TreeNode`	`TreeNode(2, None, None)`
`right_node`	`TreeNode`	`TreeNode(3, None, None)`

Функция должна возвращать итерируемый объект, содержащий результат обхода в симметричном порядке.

Заготовка кода

```
from __future__ import annotations
from typing import Iterator
```

```
class TreeNode:
    def __init__(
        self,
        val: int = 0,
        left: TreeNode | None = None,
        right: TreeNode | None = None,
    ) -> None:
        self.val = val
        self.left = left
        self.right = right

def traverse_inorder(root_node: TreeNode | None)
    -> Iterator[int]:
    # Здесь должна быть ваша реализация

one = TreeNode(1, None, None)
three = TreeNode(3, None, None)
two = TreeNode(2, one, three)
five = TreeNode(5, None, None)
seven = TreeNode(7, None, None)
six = TreeNode(6, five, seven)
four = TreeNode(4, two, six)

for node in traverse_inorder(four):
    print(node)

one = TreeNode(1, None, None)
four = TreeNode(4, None, None)
three = TreeNode(3, None, four)
two = TreeNode(2, one, three)

for node in traverse_inorder(two):
    print(node)
```

Примеры

```
Вход:
     4
   /   \
  2     6
 / \   / \
1   3 5   7
Выход: [1, 2, 3, 4, 5, 6, 7]

Вход:
  2
 / \
1   3
     \
      4
Выход: [1, 2, 3, 4]
```

Задача 48.1 (дополнительная)

Предварительные сведения

При обходе в симметричном порядке каждый узел посещается ровно один раз, поэтому временная сложность алгоритма равна $O(n)$, а пространственная – $O(h)$, где h – высота дерева.

Существует алгоритм Морриса, который обходит двоичное дерево в симметричном порядке и имеет пространственную сложность $O(1)$. Он работает следующим образом.

1. Инициализировать текущий узел корнем дерева.
2. Если у текущего узла нет левого потомка, отдать его значение и перейти к правому потомку.
3. Если у текущего узла есть левый потомок, найти его предшественника в симметричном порядке (т. е. самый правый узел в его левом поддереве).
   a. Если у предшественника нет правого потомка, то сделать его правым потомком текущий узел и перейти к левому потомку текущего узла.
   b. Если у предшественника есть правый потомок, положить его правый потомок равным `None`, отдать значение текущего узла и перейти к правому потомку текущего узла.

Задание

Реализуйте на Python алгоритм обхода Морриса. Входы и выходы должны быть такими же, как в задаче 48.

```
from __future__ import annotations
from typing import Iterator

class TreeNode:
    def __init__(
        self,
        val: int = 0,
        left: TreeNode | None = None,
        right: TreeNode | None = None,
    ) -> None:
        self.val = val
        self.left = left
        self.right = right

def morris_traverse_inorder(root_node: TreeNode | None)
    -> Iterator[int]:
```

```
    # Здесь должна быть ваша реализация

one = TreeNode(1, None, None)
three = TreeNode(3, None, None)
two = TreeNode(2, one, three)
five = TreeNode(5, None, None)
seven = TreeNode(7, None, None)
six = TreeNode(6, five, seven)
four = TreeNode(4, two, six)

for node in morris_traverse_inorder(four):
    print(node)
```

Задача 49

Предварительные сведения

Задача о подъеме по лестнице хорошо известна в информатике и математике. Требуется найти, сколькими способами можно подняться по лестнице, если на каждом шаге можно подниматься на одну или на две ступеньки.

Единственным входом является количество ступенек. Например, если ступенек три, то есть следующие способы подняться по лестнице:

- 1 ступенька, 1 ступенька, 1 ступенька;
- 1 ступенька, 2 ступеньки;
- 2 ступеньки, 1 ступенька.

Таким образом, в данной задаче ответом будет 3.

Задание

Определите функцию *solve_climbing_stairs_problem*, принимающую один параметр.

Имя	Тип	Пример входа
`total_stairs`	`int`	`4`

Функция должна возвращать число способов подняться по лестнице.

Дополнительное условие:

- в решении должно использоваться динамическое программирование, а не простая рекурсия.

Заготовка кода

```
def solve_climbing_stairs_problem(total_stairs: int) -> int:
    # Здесь должна быть ваша реализация

print(solve_climbing_stairs_problem(4))
print(solve_climbing_stairs_problem(10))
print(solve_climbing_stairs_problem(0))
```

Примеры

```
Вход: 4
Выход: 5

Вход: 10
Выход: 89

Вход: 0
Выход: 0
```

Задача 49.1 (дополнительная)

Задание

В задаче 49 вы должны были вывести число способов подняться по лестнице, но не требовалось перечислить эти способы.

Улучшите решение, добавив вывод всех способов подняться по лестнице.

Дополнительное условие:

- в решении должно использоваться динамическое программирование, а не простая рекурсия.

Заготовка кода

```
def solve_climbing_stairs_problem_with_output(total_stairs: int)
    -> list[list[int]]:
    # Здесь должна быть ваша реализация

print(solve_climbing_stairs_problem_with_output(4))
print(solve_climbing_stairs_problem_with_output(10))
print(solve_climbing_stairs_problem_with_output(0))
```

Примеры

```
Вход: 4
Выход: [[1, 1, 1, 1], [2, 1, 1], [1, 2, 1], [1, 1, 2], [2, 2]]

Вход: 10
Выход: [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [2, 1, 1, 1, 1, 1, 1, 1, 1]...]

Вход: 0
Выход: []
```

Задача 49.2 (дополнительная)

Задание

Модифицируйте решение задачи 49, при условии что за один шаг можно подниматься на 1, 2 или 3 ступеньки.

Дополнительное условие:

- в решении должно использоваться динамическое программирование, а не простая рекурсия.

Заготовка кода

```
def solve_climbing_stairs_problem_with_three_steps_allowed(
    total_stairs: int
) -> int:
    # Здесь должна быть ваша реализация

print(solve_climbing_stairs_problem_with_three_steps_allowed(4))
print(solve_climbing_stairs_problem_with_three_steps_allowed(10))
print(solve_climbing_stairs_problem_with_three_steps_allowed(0))
```

Примеры

```
Вход: 4
Выход: 7

Вход: 10
Выход: 274

Вход: 0
Выход: 0
```

Задача 50

Предварительные сведения

Задача о размене, как и задача 49, является классической в информатике. Формулируется она следующим образом: дано множество монет разного достоинства и целевая сумма, требуется найти минимальное число монет, составляющих данную сумму.

Задание

Определите функцию *solve_coin_change_problem*, принимающую два параметра.

Имя	Тип	Пример входа
`coin_values`	Список `int`	`[1, 6]`
`target_amount`	`int`	`6`

Функция должна возвращать минимальное число монет, составляющих заданную сумму, или –1, если размен невозможен.

Можно предполагать, что число монет каждого достоинства бесконечно.

Дополнительное условие:

- для получения максимально эффективного решения следует использовать динамическое программирование.

Заготовка кода

```
def solve_coin_change_problem(coin_values: list[int],
target_amount: int) -> int:
    # Здесь должна быть ваша реализация

print(solve_coin_change_problem([1, 6], 6))
print(solve_coin_change_problem([1, 2], 6))
print(solve_coin_change_problem([2, 1], 13))
```

Примеры

```
Входы:
 - coin_values: [1, 6]
 - target_amount: 6
Выход: 1
```

```
Входы:
 - coin_values: [1, 2]
 - target_amount: 6
Выход: 3

Входы:
 - coin_values: [2, 1]
 - target_amount: 13
Выход: 7
```

Шуточные задачи

Задачи в этом разделе построены несколько иначе, обычно в них используются внешние библиотеки, с которыми вы можете быть или не быть знакомы. Если у вас нет опыта использования какой-то библиотеки, не пропускайте задачу! Это отличный шанс научиться чему-то новому.

Обратный поиск в DNS

Предварительные сведения

DNS (Domain Name System – система доменных имен) – это система, которая транслирует доменные имена в понятном человеку формате, например example.com, в IP-адреса, которыми компьютеры пользуются для идентификации друг друга в интернете. Система работает как справочная служба, которая дает пользователям возможность обращаться к сайтам и другим сетевым ресурсам путем ввода доменного имени в своем браузере. Затем это имя преобразуется в IP-адрес DNS-сервером.

Задание

Определите функцию *reverse_dns_lookup*, принимающую один параметр:

Имя	Тип	Пример входа
`ip_address`	`str`	`"8.8.8.8"`

Функция должна возвращать доменное имя, которое транслируется в указанный IP-адрес, для чего следует использовать DNS-записи типа PTR. Функция должна возвращать `None`, если для указанного IP-адреса нет PTR-записи.

Воспользуйтесь функцией *gethostbyaddr* из библиотеки `socket`.

Заготовка кода

```
import socket

def reverse_dns_lookup(ip_address: str) -> str:
    # Здесь должна быть ваша реализация

print(reverse_dns_lookup("8.8.8.8"))
print(reverse_dns_lookup("208.67.222.222"))
print(reverse_dns_lookup("1.1.1.1"))
```

Примеры

Примечание: возможно, хотя и маловероятно, что в тот момент, когда вы будете выполнять код, результат изменится. Но, скорее всего, он по-прежнему будет правилен.

```
Вход:   "8.8.8.8"
Выход:  "dns.google"

Вход:   "208.67.222.222"
Выход:  "dns.umrella.com"

Вход:   "1.1.1.1"
Выход:  "one.one.one.one"
```

Матрешки

Предварительные сведения

Матрешка представляет собой набор деревянных кукол уменьшающегося размера, вставленных одна в другую. Самая большая кукла открывается, в ней находится кукла поменьше, в ней еще меньше, и так далее, пока куклы не кончатся.

Задание

Определите функцию *unpack_dolls*, принимающую один параметр:

Имя	Тип	Пример входа
`Doll`	`RussianDoll`	`RussianDoll(4, 3, purple")`

Параметр имеет тип `RussianDoll`, это простой класс с тремя свойствами:

- Size – размер куклы;
- Colour – цвет куклы;
- Child doll – еще один объект `RussianDoll`, находящийся внутри данной куклы, или `None`.

Класс `RussianDoll` включен в заготовку кода ниже. Функция *unpack_dolls* должна выводить данные наподобие показанных в примере ниже.

Пример

Пусть имеются следующие матрешки:

Размер куклы	Цвет	Размер внутренней куклы
5	Серый	4
4	Фиолетовый	3
3	Зеленый	2
2	Синий	1
1	Красный	None

Тогда программа должна вывести на консоль:

```
Unpacking a grey doll of size: 5 with 4 nested dolls inside.
Unpacking a purple doll of size: 4 with 3 nested dolls inside.
Unpacking a green doll of size: 3 with 2 nested dolls inside.
Unpacking a blue doll of size: 2 with 1 nested dolls inside.
```

```
Unpacking a red doll of size: 1 with 0 nested dolls inside.
Total number of dolls in the set: 5
```

Заготовка кода

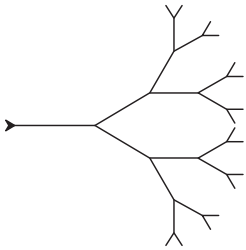
```
from __future__ import annotations

class RussianDoll:
    def __init__(
            self,
            size: int,
            colour: str,
            child_doll: RussianDoll | None = None
    ) -> None:

        self.size = size
        self.colour = colour
        self.child_doll = child_doll

def unpack_dolls(doll: RussianDoll) -> int:
    # Здесь должна быть ваша реализация

doll_size_one = RussianDoll(1, "red", None)
doll_size_two = RussianDoll(2, "blue", doll_size_one)
doll_size_three = RussianDoll(3, "green", doll_size_two)
doll_size_four = RussianDoll(4, "purple", doll_size_three)
doll_size_five = RussianDoll(5, "grey", doll_size_four)
unpack_dolls(doll_size_five)
```

Фрактальное дерево

Предварительные сведения

Фрактальным деревом называется древовидная структура, подобная самой себе на различных масштабах. Она создается путем рекурсивного применения правил или алгоритмов для генерирования разветвлений, которые выглядят так же, как у настоящих деревьев.

Ниже приведен пример фрактального дерева:



Задание

Определите функцию *draw_tree*, принимающую один параметр:

Имя	Тип	Пример входа
`depth`	`int`	`5`

Функция должна отображать фрактальное дерево.

Вы должны использовать библиотеку черепашьей графики для Python. На случай, если вы с ней незнакомы, скажем, что это простой и удобный способ создавать графические изображения и визуализации с помощью виртуальной «черепашки». Черепашка – это курсор, которым можно управлять для рисования фигур и линий на холсте.

Заготовка кода

```
import turtle

def draw_tree(depth: int) -> None:
    # Здесь должна быть ваша реализация

draw_tree(5)
turtle.exitonclick()
```

Пример

Глубина показанного выше дерева была равна 5. Ваше дерево глубины 5 должно выглядеть так же.

Пинг-понг

Задание

Напишите простую игру в пинг-понг для двух игроков с помощью библиотеки pygame. В игре должны быть две ракетки, один мячик и система подсчета очков.

Примечание: такого рода задачи оказываются интереснее, если отпустить на волю воображение, поэтому я намеренно сделал описание расплывчатым. Тут не может быть правильных или неправильных ответов, коль скоро конечная программа напоминает какую-то форму игры в пинг-понг!

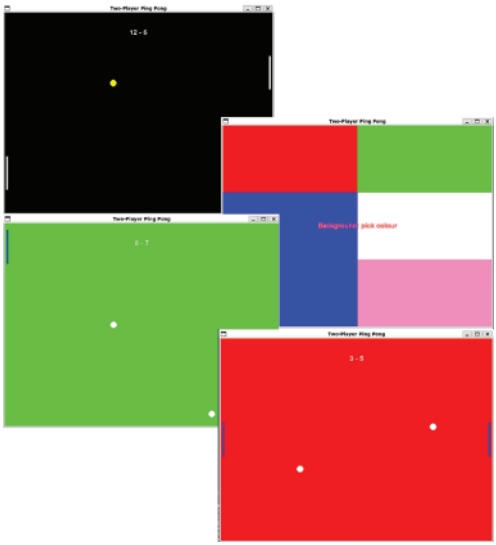
Дополнительные условия:

- разрешите игрокам выбирать желаемый цвет стола, мяча и ракеток (*еще одно усложнение: реализуйте визуальное средство выбора цвета*);
- заставьте мяч менять цвет каждую секунду;
- разрешите одновременно играть произвольным числом мячей.

Указание: дополнительные условия проще реализовать с применением объектно ориентированного программирования (... подумайте о нескольких мячах в игре).

Примеры

Поскольку это творческая задача, все решения будут различны – и это хорошо! Справа показано несколько примеров возможных решений.



Средство рисования

Задание

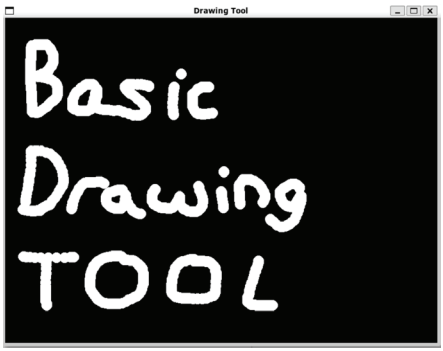
Напишите простую программу рисования, воспользовавшись библиотекой pygame. Отобразите черный холст, щелкая по которому, пользователь сможет рисовать цветной кистью.

Имея это приложение, мы располагаем неплохой базой для добавления функциональности. Вот несколько дополнительных идей:

- реализуйте боковую панель, которая позволяет:
- выбрать другой цвет кисти;
- очистить холст;
- сохранить рисунок на диск;
- изменить толщину кисти;
- создайте средство проведения линий, чтобы пользователь мог выбирать между «рисованием кистью» и «инструментом рисования линий».

Но не останавливайтесь на этом! Это только идеи для начала, а вы можете развивать свое приложение в любом направлении!

Примеры



Серьезные задачи: указания

Задача 1

Вам нужно воспользоваться циклом for для перебора всех строк в списке `input_strs`. В цикле проверяйте, содержит ли текущее слово букву «a», это можно сделать в предложении if.

Чтобы проверить, содержит ли строка некоторую букву, можно воспользоваться оператором `in`, например:

- *«a» in «banana» = True*;
- *«a» in «pop» = False*.

Задача 2

Первый шаг решения – сложить два числа.

Второй шаг – проверить, что получившаяся сумма меньше 50. Проверить это можно с помощью предложения if. Если условие if равно true (сумма меньше 50), то мы возвращаем сумму. В противном случае возвращаем `None`.

Задача 3

Решение можно разбить на несколько шагов.

Первый шаг – отфильтровать нечетные числа; для этого можно воспользоваться оператором деления по модулю (%), который возвращает остаток от деления, например:

- 5 % 2 = 1;
- 4 % 2 = 0.

Мы можем применить этот оператор в паре со списковым включением, чтобы создать новый список, содержащий только четные числа. Для тех, кто незнаком со списковым включением, скажем, что это следующая синтаксическая конструкция:

[<имя переменной> for <имя переменной> in <список> if <условие>]

Списковое включение `[my_number for my_number in [1, 2, 3] if my_number % 3 == 0]` даст список [3], потому что 1 % 3 = 1 и 2 % 3 = 2, так что в обоих случаях условие равно False.

Затем можно воспользоваться встроенной в Python функцией `sum`, которая складывает все элементы списка.

Итак, если мы сначала создадим список, содержащий только четные элементы, а затем просуммируем его и вернем результат… то мы решим задачу!

Задача 4

Задание

Для решения этой задачи применяется комбинация методов, рекомендованных для задач 1 и 3.

Нам нужно использовать условное списковое включение для перебора всех символов входной строки. Условие должно проверять, является ли текущий символ гласной буквой (указание: для этого можно использовать оператор `in`, упомянутый в задаче 1). Получив результирующий список, мы можем преобразовать его в строку с помощью встроенной функции `join`.

Задача 5

Для начала спросим себя: если не смотреть на входные строки, то какую самую длинную строку мы можем предложить? Пустую строку! Тогда идея решения выглядит так:

- положить «самую длинную» строку равной пустой строке;
- обойти все входные строки, и если текущая строка длиннее «самой длинной», то положить «самую длинную» строку равной текущей;
- в конце программы «самая длинная» строка действительно будет самой длинной!

Для нахождения длины строки можете воспользоваться встроенной функцией `len`.

Задача 6

Я сознательно опускаю указания для этой задачи. Посоветую только взглянуть, как предлагалось решать задачи 3, 4 и 5.

Задача 7

Задание

В решении следует использовать срезы Python. Синтаксически срез обозначается квадратными скобками после списка, внутри которых указываются параметры среза: *начало:конец:шаг*.

Например, если взять список [1,2,3,4,5,6] и срез [0:4:2], то получим список [1, 3], поскольку параметры среза означают, что нужно начать с индекса 0 (включительно) и идти до индекса 4 (исключительно) с шагом 2.

В позиции списка с индексом 0 находится число 1. Сделав шаг длиной 2, мы попадаем в позицию с индексом 2, где находится число 3. А сделав еще один шаг длиной 2, мы перешагиваем конечную позицию и должны остановиться.

Итак, какие параметры среза задать, чтобы обратить список? Указание: шаг должен быть отрицательным!

Задача 8

Для этой задачи вам понадобится использовать списковое включение и встроенную функцию `isinstance`, которая работает следующим образом:

- если в вызове `isinstance(variable, type)` тип переменной `variable` равен `type`, то возвращается `True`, иначе `False`, например:

```
isinstance("a string", str) = True
isinstance("a string", int) = False
```

Задача 9

Решение должно начинаться созданием пустого списка, в котором будет храниться результат.

Затем нужно перебрать все символы входной строки.

1. Если символ – не пробел, то используем `morse_dict` для получения его кода Морзе и добавляем результат в список.
2. Если символ – пробел, то добавляем в список знак косой черты.

Построив список, мы можем преобразовать его в строку, воспользовавшись той же техникой, что в задаче 4.

Задача 10

Разобьем решение на два шага.

1. В списке есть хотя бы два числа? Если нет, возвращаем `None`.
2. Найти второе по величине число:
   a. Найти и удалить из списка максимальное число (указание: воспользуйтесь встроенной функцией `max`).
   b. Теперь мы знаем, что наибольшее число в оставшемся списке – это второе по величине число в исходном списке. А значит, можно еще раз воспользоваться функцией `max` для нахождения наибольшего числа и вернуть результат.

Задача 11

Задание

Эту задачу можно решить несколькими способами, но самый простой – воспользоваться f-строками. В Python f-строки (форматные строковые литералы) дают краткий и удобный способ вкладывать выражения в строковые литералы, позволяя тем самым создавать строки с динамическим содержимым. Они появились в версии Python 3.6 как способ упростить форматирование строк по сравнению с такими более старыми средствами, как оператор `%` или метод `.format()`.

Почитайте документацию по f-строкам – там много интересного!

Задачи 12 и 12.1

В этом решении мы воспользуемся встроенными в Python функциями `ord` и `chr`.

- Функция `ord` принимает один символ и возвращает его ASCII-код.
- Функция `chr` принимает ASCII-код и возвращает символ с таким кодом.

Эти функции вместе с рассмотренными в предыдущих задачах приемами укажут путь к решению этой задачи!

Задача 13

Это первое решение, где в голову может прийти мысль о написании вспомогательной функции, т. е. такой, которая будет вызываться из функции `filter_strings_with_vowels`.

Предложение: определите вторую функцию `has_vowel`, которая принимает входную строку и возвращает `True`, если она содержит гласную букву, и `False` в противном случае.

Затем можно будет использовать условное списковое включение, которое будет нужным образом фильтровать строки по условию `has_vowel`.

Задача 14

Используйте указания к задаче 7, касающиеся срезов.

Задача 15

В вашем решении следует использовать списковое включение и срезы Python. Может помочь решение задачи 7...

Задача 16

Решение состоит из нескольких шагов.

1. Перебор всех входных строк.
2. Перебор всех символов в каждой входной строке. При этом нужно:
   a) позаботиться о нечувствительности символа к регистру (указание: воспользуйтесь встроенной в Python функцией `upper`);
   b) проверить, равен ли символ одной из букв P, Y, T, H, O или N;
   c) при необходимости заменить символ на X.
3. Вернуть список строк с подвергнутыми цензуре символами.

В основу решения можно было бы положить списковое включение, ключевое слово `in` и функцию `join`.

Задача 17

В решении можно было бы использовать три переменные, которые пробегают по списку и проверяют, что каждые три соседних значения различны. Выглядеть это могло бы так:

```
input_str = "abcdef"
a = input_str[0] = "a";
b = input_str[1] = "b";
c = input_str[2] = "c";

Правда ли, что a, b и c различны?
- Если да, увеличить все индексы на 1 и продолжить
- Если нет, например если a == b, то строка несчастливая.

a = input_str[1] = "b";
b = input_str[2] = "c";
c = input_str[3] = "d";

Правда ли, что a, b и c различны?
- Если да, увеличить все индексы на 1 и продолжить
- Если нет, например если a == b, то строка несчастливая.

... и так далее ...
```

Но так зашивать код в программу непрактично. А как сделать это эффективно на Python?

Задача 18

Это первая рекурсивная задача, поэтому на случай, если эта концепция вам незнакома, далее следует краткое введение в тему.

Рекурсия – это техника программирования, при которой функция вызывает саму себя. Как будто функция смотрит в зеркало и просит себя помочь в решении той же задачи, но меньшего размера. Так продолжается до тех пор, пока задача не станет настолько малой, что ее можно решить непосредственно.

Приведем простой пример рекурсии:

```
def countdown(n):
    if (n <= 0):
        print("Go!")
    else:
        print(n)
        countdown(n - 1)

countdown(5)
```

Функция *countdown* ведет обратный отсчет от *n* до 0. Если *n* меньше или равно 0, то она печатает «Go!». В противном случае она печатает текущее значение *n*, а затем вызывает себя же с аргументом *n* – 1. Это продолжается до тех пор, пока *n* не станет равным 0 или отрицательным, и в этот момент рекурсия останавливается.

Обычно у рекурсии есть базовый случай, когда функция перестает вызывать себя. В примере выше это часть `if (n <= 0)`. Если базовый случай еще не наступил, то рекурсия продолжается.

А теперь вспомним о задаче 18...

- Чего мы хотим от рекурсии?
- Как насчет того, чтобы делить (нацело) `input_num` на 10 при каждом рекурсивном вызове функции?
  - 1234 // 10 = 123
  - 123 // 10 = 12
  - 12 // 10 = 1
  - 1 // 10 = 0
- Хорошо, но как мы собираемся подсчитывать число цифр?
  - Заметим, что после каждого деления из `input_num` исчезает одна цифра. Поэтому будем при каждом делении прибавлять 1 к счетчику, пока не дойдем до базового случая, и тогда вернем текущий счетчик.

Поделившись этой идеей, я оставляю вам для самостоятельного изучения второй рекурсивный пример – нахождение длины строки.

```
def get_length_of_str(input_str):
    # Базовый случай
    if (len(input_str) == 0:
        return 0

    # На шаге рекурсии к счетчику прибавляется 1, а из конца входной строки
    # удаляется один символ
    return 1 + get_length_of_str(input_str[:-1])
```

Задача 19

Разобьем решение на четыре части.

1. Обойти в цикле все строки и проверить, выиграл ли кто-нибудь. Например, если все три элемента в верхней строке равны X, то вернуть «X».
2. Обойти в цикле все столбцы и проверить, выиграл ли кто-нибудь. Например, если все три элемента в среднем столбце равны O, то вернуть «O».
3. Проверить диагонали. Например, если левый верхний, средний и правый нижний элементы равны O, то вернуть «O».
4. Если ни одно из этих условий не выполнено, то вернуть `None`.

Задача 20

Над этой задачей надо немного подумать, но решить ее можно с помощью одного цикла и одного предложения печати.

- Цикл будет представлять каждый уровень треугольника, начиная с 1 и до `number_of_levels + 1`.
- Предложение печати должно:
  - напечатать нужное число пробелов;
  - напечатать нужное число символов.

Сколько пробелов нужно напечатать на каждом уровне?

- Нужно напечатать `number_of_levels - current_level` пробелов. Например, если уровней 5, а мы находимся на уровне 1, то пробелов должно быть 4.

Сколько символов нужно напечатать на каждом уровне?

- Число символов равно `current_level * 2 - 1`. Например, если мы находимся на втором уровне, то нужно напечатать три символа: 2 * 2 - 1 = 3.

Задача 21

После задачи 18 мы знаем, что у рекурсии должен быть базовый случай и шаг рекурсии. В этой связи поразмыслите о следующем.

- Шаг рекурсии:
  - Как вычисляется последовательность чисел Фибоначчи?
  - Как сделать так, чтобы на шаге рекурсии вычислялась сумма двух предыдущих чисел Фибоначчи?
- Базовый случай: если на шаге рекурсии всегда вычисляется сумма двух предыдущих чисел Фибоначчи, то в какой момент следует остановиться и вернуть результат?
  - Указание: какие члены последовательности чисел Фибоначчи невозможно вычислить как сумму двух предыдущих членов?

Для пущей ясности рассмотрим, как будет выглядеть стек в процессе рекурсии:

```
fibonacci(4)

# Шаг рекурсии
= fibonacci(3) + fibonacci(2)

# Шаги рекурсии при вычислении fibonacci(3) и fibonacci(2)
= fibonacci(2) + fibonacci(1) + fibonacci(1) + fibonacci(0)

# Шаг рекурсии при вычислении fibonacci(2)
# Базовыми случаями являются fibonacci(1) и fibonacci(0)
= fibonacci(1) + fibonacci(0) + 1 + 1 + 0

# fibonacci(1) и fibonacci(0) являются базовыми случаями
= 1 + 0 + 1 + 1 + 0

= 3
```

Задача 22

Надеюсь, вы уже освоились с рекурсивными решениями. Ключ к решению этой задачи – шаг рекурсии (базовый случай похож на задачу о числах Фибоначчи, хотя и не совсем такой же).

Что нужно сделать на шаге рекурсии?

1. Вычислить 1, поделенную на текущее число (n).
2. Вычислить гармоническую сумму для всех чисел от n – 1 до 1 (указание: используйте рекурсию).

Сложив результаты шагов 1 и 2, мы получим искомый шаг рекурсии!

Задача 23

Как всегда, ключ к решению задачи – разбить ее на части.

1. Инициализировать результирующую строку.
2. Обойти все входные строки и вычислить операцию XOR.
   a. Сначала нужно найти длину самой короткой строки, потому что избыточные символы более длинных строк отбрасываются.
   b. Затем в цикле от 0 до длины самой короткой строки применяем операцию XOR к `input_a[currentindex]` и `input_b[current index]`.
   c. В зависимости от результата XOR дописываем в результирующую строку 0 или 1.
3. И наконец, возвращаем результат.

Задача 24

При решении этой задачи следует поступать так же, как при решении задачи 23. Инициализировать переменную для хранения результата, выполнить цикл от 0 до длины самого короткого списка, а затем дописать кортеж, содержащий пару, в конец результата.

Задача 25

Ключ к решению задачи – тот факт, что все части равенства разделены пробелами. Это значит, что мы можем воспользоваться встроенной в Python функцией `split` для разбиения равенства на части.

Имея части равенства, остается просто выполнить сложение или вычитание и проверить, что результаты равны.

Задача 26

В этой задаче запрещено использовать циклы, поэтому воспользуемся срезом списка. Срез должен выглядеть так:

```
input_list[rotate_amount:] + input_list[:rotate_amount]
```

При этом создается иллюзия циклического сдвига списка влево. Например:

```
Входы:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 2

input_list[rotate_amount:] = [3, 4, 5]
input_list[rotate_amount:] = [1, 2]
input_list[rotate_amount:]-
    + input_list[rotate_amount:] = [3, 4, 5, 1, 2]
```

Остается понять, чему должно быть равно значение `rotate_amount`. Вспомните, что функция должна уметь обрабатывать сдвиг на величину, большую длины списка. Указание: в решении можно использовать оператор деления по модулю (%).

Задача 27

Эту задачу можно решить с помощью одного условного спискового включения. В условии следует использовать такой факт Python:

- вычисление `if 1` дает `True`;
- вычисление `if 0` дает `False`.

В списковом включении следует воспользоваться встроенной в Python функцией `enumerate`. На всякий случай скажу, что эта функция позволяет добавить счетчик в цикл по элементам, например:

```
my_list = ["a", "b", "c"]
for counter, value = enumerate(my_list):
    print(f"{counter}, {value}")
```

```
Выход:
0, a
1, b
2, c
```

Подумайте, как, объединив все это, прийти к решению?

Задача 28

Разобьем задачу на шаги.

1. Инициализировать две переменные для подсчета числа пиков и впадин.
2. Пробежаться по списку с индекса 1 (не 0) до `len(price_action) - 1`. Будем обозначать индекс `i`.
3. Зная `price_action`, `i - 1`, `i` и `i + 1`, мы можем определить, является `i`-й элемент пиком или впадиной (как именно, догадайтесь сами).
4. При обнаружении пика или впадины нужно увеличить соответствующий счетчик.
5. По выходе из цикла возвращаем счетчики.

Задача 29

В заготовке кода уже имеется словарь кода для перестукивания, который мы будем использовать в решении, поэтому нужно подумать, как преобразовать букву в код и наоборот. Чтобы преобразовать букву `"a"` в код для перестукивания, нужно найти `tap_code_map["a"]` – это будет `". ."`. Но в обратную сторону – для преобразования `". ."` в `"a"` – так не получится.

Дабы справиться и с этой задачей, мы должны инвертировать словарь, так чтобы ключи стали значениями, а значения – ключами. Это первый шаг решения. Указание: подумайте, как воспользоваться списковым включением.

Второй шаг – завести предложение if, которое будет проверять, является ли `input_code` пустым, и если да, то возвращать пустую строку.

Теперь подумаем, как обработать настоящий код.

1. Разбить `input_code` на «слова кода», памятуя о том, что слова разделены тремя пробелами.
2. Для каждого слова кода:
   a) разбить слово на отдельные коды букв, памятуя о том, что буквы разделены двумя пробелами;
   b) для каждого кода буквы в слове:
      – найти код в инвертированном словаре и вернуть соответствующее значение буквы;
   c) соединить все буквы слова, получив тем самым строку, представляющую слово.
3. Соединить все слова, воссоздав понятное человеку предложение.

Задача 30

Тут не нужно никакой зауми. Понадобится только список результатов, в который будут добавляться найденные комбинации и три цикла `for`. Детали оставляю вам.

Задача 31

Погуглите «Python *args» – вы получите все, что нужно для решения! Эта задача служит для разогрева перед задачей 31.1.

Задача 31.1 (дополнительная)

Вернуть список…

Попробуйте реализовать все шаги показанного ниже наброска решения.

1. Если хотя бы один входной список пуст, вернуть пустой список.
2. Найти самый короткий входной список, обозначим его длину `shortest_length`.
3. Инициализировать результирующий объект – список пустых кортежей в количестве, равном `shortest_length`.
4. В цикле по индексу `i` от 0 до `shortest_length` выполнить:
   а) написать внутренний цикл, перебирающий все входные списки:
      – дописать `input_list[i]` в соответствующий кортеж результирующего объекта.
5. Вернуть результирующий объект.

Вернуть итератор…

Я не рекомендую пробовать эту версию, не доведя до конца вариант с возвратом списка. Но если вы уже это сделали, то ниже приведем псевдокод, который поможет справиться с версией для итератора.

```
def my_zip_one(*input_lists: list[Any])
    -> Iterator[tuple[Any, ...]]:

    sentinel = object()

    input_lists_iterators
        = список итераторов для каждого входного списка в input_lists

    while input_lists_iterators не пуст:
        zipped_result = пустой список

        for each input_list_iterator in input_lists_iterators:

            получить следующий элемент от input_lists_iterator;
            если следующего элемента нет, то взять значение sentinel.

            # Проверить, исчерпан ли входной список
            if элемент совпадает с sentinel:
                выйти из цикла

            добавить элемент в zipped_result

        yield zipped_result как Tuple
```

Задача 32

Если вы застряли на этой задаче, подумайте о том, как можно было бы использовать множества. Помните, что множества не упорядочены, что доказывает такой пример:

```
>>> set("hello") == set ("oelhl")
True
```

Могли бы вы сделать нечто подобное для обнаружения букв слова «python»?

Задача 33

Можно было воспользоваться вспомогательной функцией в сочетании с условным списковым включением – по аналогии с тем, что было сделано в решении задачи 13.

Вспомогательная функция должна выяснить, является ли число простым, и в зависимости от этого вернуть `True` или `False`. Для этого нужно проверить два условия:

1. Верно ли, что число меньше 2?
2. Делится ли это число на какое-то другое?

Задача 34

Первая подсказка для решения этой задачи: можно ли как-то использовать ASCII? Быть может, какие-то из функций, рассмотренных в задачах 12 и 12.1?

Возможно, поможет также следующий псевдокод:

```
def rot13(input_str: str) -> str:
    result = ""

    for each char in input_str
        if char буква
            if char заглавная буква
                a_code = char_to_ascii("A")
            else
                a_code = char_to_ascii("a")

            char = ascii_to_char((
                 char_toascii(char) - a_code + 13
            ) % 26 + a_code)

        добавить char в конец result += char

    return result
```

Задача 35

Эту задачу можно решить с использованием стека следующим образом:

1. Инициализировать пустой стек.
2. Перебрать все символы входной строки.
3. Встретив открывающую скобку, поместить ее в стек.
4. Когда встретится закрывающая скобка:
   а) если стек не пуст:
      – извлечь открывающую скобку из стека;
      – если стек опустел, то обнаружена полная группа. Добавить эту группу в выходную строку, при этом в качестве побочного эффекта будут убраны все пробелы.
5. Вернуть результирующую строку.

Задача 36

Первый совет при решении этой задачи: разобраться, как перемножаются матрицы. Если вы плаваете в этом вопросе, то решить задачу будет труднее. В сети есть много ресурсов на тему умножения матриц.

Поняв, как работает умножение матриц, можете взглянуть на показанный ниже псевдокод. Помните, что ваше решение необязательно должно быть таким, но это один из возможных способов.

```
def matrix_multiply(
    left_matrix: list[list[int]], right_matrix: list[list[int]]
) -> list[list[int]]:

    num_left_rows = get_number_of_rows(left_matrix)
    num_left_cols = get_number_of_columns(left_matrix)
    num_right_cols = get_number_of_columns(right_matrix)

    if num_left_rows != num_right_cols:
        return None

    result_matrix
        = create_empty_matrix(num_left_rows, num_right_cols)

    for i = 0 to num_left_rows - 1:
        for k = 0 to num_left_cols - 1:
            for j = 0 to num_right_cols - 1:
                result_matrix[i][j] +=
                    left_matrix[i][k] * right_matrix[k][j]

    return result_matrix
```

Задача 37

Главная подсказка в этой задаче – тот факт, что ее можно решить рекурсивно. Вспомните другие задачи, которые мы уже решили рекурсивно, и подумайте о том, как приспособить найденные тогда решения к этой задаче. Каким должен быть базовый случай? А каким – шаг рекурсии?

Дополнительный совет: можете воспользоваться оператором деления по модулю (%).

Задача 38

Для решения можно воспользоваться вложенными циклами, как показано в псевдокоде ниже:

```
def find_pairs_summing_to_target(
    input_nums: list[int], target: int
) -> list[tuple[int, int]]:

    pairs = create_empty_list()

    for left_idx = 0 to length(input_nums) - 1:
        left_num = input_nums[left_idx]

        for right_idx = left_idx + 1 to length(input_nums) - 1:
            right_num = input_nums[right_idx]

            if left_num + right_num == target:
               pairs.append((left_num, right_num))

    return pairs
```

Задача 38.1 (дополнительная)

Вот несколько советов, как решить эту задачу, не раскрывающих слишком много:

- в решении следует использовать два множества:
- первое будет содержать результирующие пары;
- второе будет содержать числа, необходимые для составления пары (в нашем примере они называются `seen`);
- в решении должен использоваться один цикл for;
- в решении должно использоваться одно предложение if-else.

Посмотрим, как могла бы выглядеть обработка:

```
target = 10
input_nums = [1, 2, 9]
pairs = [] # пары чисел, в сумме дающих target
seen = [] # числа, необходимые для составления пары

# В цикле обойти список input_nums, обозначив текущий элемент input_num...

# Число target - input_num находится в seen?
10 - 1 in [] = False, поэтому добавляем 1 в seen

target = 10
input_nums = 2
pairs = []
seen = [1]

# Число target - input_num находится в seen?
10 - 2 in [1] = False, поэтому добавляем 2 в seen

target = 10
input_nums = 9
pairs = []
seen = [1, 2]

# Число target - input_num находится в seen?
10 - 9 in [1, 2] = True, поэтому добавляем (9,1) в список pairs
pairs = [(9,1)]
```

Задача 39

Главная подсказка к этой задаче – искать рекурсивное решение, но это труднее, чем в других задачах. Вам понадобится один базовый случай и два шага рекурсии с печатью между ними.

Но есть и хорошая новость – больше ничего не нужно. Один базовый случай, за ним рекурсивный вызов, печать и еще один рекурсивный вызов. Проработку деталей оставляю вам!

Задача 40

Сортировку вставками поначалу понять трудно, но стоит в ней разобраться, как все проясняется! Вот несколько указаний по решению этой задачи:

- начните с обхода входного списка чисел;
- произведите концептуальное разбиение списка на две части: отсортированную и неотсортированную. Первоначально отсортированная часть состоит только из одного элемента, а все остальное не отсортировано;
- на каждой итерации выберите какое-то значение из неотсортированной части списка. Это значение будет вставлено в правильную позицию в отсортированной части;
- сравните выбранное значение с элементами в отсортированной части списка. Переместите те элементы отсортированной части, которые больше выбранного значения, чтобы освободить место для вставки;
- после того как правильная позиции в отсортированной части найдена, вставьте в нее выбранное значение.

По завершении всех итераций список будет отсортирован!

Задача 41

Ключ к этой задаче – придумать, как эффективно использовать отображение *roman_map*... Можете ли вы обойти его в цикле? Быть может, стоит использовать встроенную в Python функцию *sorted*?

Задача 41.1 (дополнительная)

Начнем с функции *roman_to_int*. Решение могло бы выглядеть так:

- инициализировать переменную `result` значением 0;
- обойти все римские цифры в `input_str` и преобразовать их в числа с использованием *roman_map*;
- прибавить число к переменной `result` и продолжать, пока строка `input_str` не будет исчерпана.

Однако тут имеется проблема: как быть с правилом вычитания? Например, если `input_str` равна «IV», то результат должен быть равен 4, однако наш алгоритм вернет 6. Предлагаю подсказку:

- если вычитание не используется, то римские цифры будут следовать в порядке убывания. Например, 16 = «XVI», здесь X больше V, а V больше I. Можно ли воспользоваться этой информацией для распознавания вычитания и поступить соответственно?
- пример с вычитанием: 19 = «XIX», здесь X больше I, но I меньше X.

Дойдя до написания функции *int_roman_converter*, можете воспользоваться следующим псевдокодом:

```
def int_roman_converter(to_convert: str | int) -> int | str:
    # римское число в int
    if to_convert является строкой:
        return roman_to_int(to_convert)

    # int в римское число
    if to_convert является int:
        return int_to_roman(to_convert)

    return None
```

Задача 42

Для решения этой задачи понадобится использовать поразрядные операции в Python, а именно:

```
# & (И - результат равен 1, только если левый и правый операнды равны 1)
1 & 1 = 1
1 & 0 = 0
0 & 1 = 0
0 & 0 = 1

# ^ (ИСКЛЮЧАЮЩЕЕ ИЛИ - результат равен 1, только если левый или правый операнды равны 1,
# но не оба сразу)
1 ^ 1 = 0
1 ^ 0 = 1
0 ^ 1 = 1
0 ^ 0 = 0

# << (сдвиг влево - по существу, умножение числа на 2)
0b1 << 1 = 0b10
0b10 << 1 = 0b100
0b101 << 1 = 0b1010
```

Попытайтесь решить задачу самостоятельно, а если никак не получается, то воспользуйтесь следующим псевдокодом:

```
def bitwise_add(num_one: int, num_two: int) -> int:
    while num_two не равно 0:
        carry = какая-то операция с применением И...
        num_one = какая-то операция с применением ИСКЛЮЧАЮЩЕГО ИЛИ...
        num_two = какая-то операция с применением сдвига влево...

    return num_one
```

Задача 43

Первый шаг к решению этой задачи – понять, как работает двоичный поиск. В помощь вам множество сетевых ресурсов, но постарайтесь все-таки не подглядывать в решения на Python!

Разобравшись с основами работы алгоритма, можете воспользоваться следующим псевдокодом:

```
def binary_search(sorted_list: list[int], value_to_find: int) -> int:

    low = 0
    high = длина sorted_list - 1
    mid = 0

    while low меньше или равно high:
        mid = среднее high и low, округленное с недостатком

        если value_to_find больше средней точки списка, то игнорировать
        левую половину, изменив переменную low

        если value_to_find меньше средней точки списка, то игнорировать
        правую половину, изменив переменную high

        иначе мы нашли value_to_find в позиции sorted_list[mid]

    # если мы оказались в этой точке, значит, искомый элемент отсутствует
    return -1
```

Задача 44

Как и в случае двоичного поиска выше, прежде всего следует убедиться, что вы полностью понимаете, как работает алгоритм быстрой сортировки (это не просто и может потребовать некоторого времени)! Ну а после этого вашему вниманию предлагается следующий псевдокод:

```
def quicksort(input_list: list[int], low: int, high: int) -> list[int]:
    if low < high:
        # Найти индекс опорного элемента
        pivot_idx = partition(input_list, low, high)

        рекурсивно отсортировать оба подсписка, два раза вызвав quicksort:
        - один раз с low = low и high = pivot_idx - 1
        - второй раз с low = pivot_idx + 1 и high = high

def partition(input_list: list[int], low: int, high: int) -> int:
    pivot = input_list[high]
    pivot_idx = low - 1

    for current_idx = low to  high - 1:
        if текущий элемент меньше или равен pivot:
           - сдвинуть индекс опорного элемента вправо
           - swap(input_list[pivot_idx], input_list[current_idx])

    swap(input_list[pivot_idx + 1], input_list[high])

    return pivot_idx + 1
```

Задача 45

Для решения задачи 45 можно использовать рекурсию. Определим вспомогательную функцию, принимающую три параметра:

1) `items` – то же, что `items` в функции `solve_knapsack_problem`;
2) `index` – индекс предмета (в `items`), являющегося кандидатом на укладывание в рюкзак;
3) `remaining_capacity` – сколько места осталось в рюкзаке.

У функции должен быть базовый случай и три шага рекурсии.

- Базовый случай: мы рассмотрели все предметы или вместимость равна нулю.
- Шаги рекурсии:
  - если вес текущего предмета больше оставшейся вместимости, то его нельзя положить в рюкзак;
  - если текущий предмет можно положить, то:
    - положить его и рекурсивно проверить оставшуюся вместимость;
    - не класть его и рекурсивно проверить оставшуюся вместимость.

Таким образом, мы рассмотрим все возможности для всех предметов и вернем правильный ответ.

Задача 45.1 (дополнительная)

В задаче 45.1 требуется улучшить решение задачи 45, решив ее за время $O(nW)$, где n – число предметов, а W – вместимость рюкзака.

Для этого нам потребуется динамическое программирование. По сути дела, это просто оптимизация рекурсии, когда мы сохраняем результаты подзадач. Сохранение результата означает, что нам не придется повторно вычислять его, когда он снова понадобится.

Рассмотрим псевдокод, который позволит вам довести решение до конца...

```
def solve_knapsack_problem_bonus(
    items: list[tuple[int, int]], knapsack_capacity: int
) -> int:

    # Двумерная матрица нулей для хранения максимальных значений.
    # число строк = len(items) + 1, число столбцов = knapsack_capacity + 1
    max_value_matrix = [[0, 0, ...], ...]
    ]

    for item_idx = 1 to len(items) + 1:
        for weight = 1 to knapsack_capacity + 1:

        # Проверить, превосходит ли вес текущего предмета оставшуюся
        # вместимость
        if items[item_idx - 1][0] > weight:
            # Обновить значения в матрице max_values
            max_value_matrix[item_idx][weight]
                = max_value_matrix[item_idx - 1][weight                ]
        else:
            # Вычислить максимальное значение, рассмотрев два случая:

            # 1. Исключая текущий предмет
            exclude_current = ...

            # 2. Включая текущий предмет
            include_current = ...

            # Выбрать максимум из двух случаев
            max_value_matrix[item_idx][weight]
                = max(exclude_current, include_current)

    # Последнее значение в матрице равно максимально достижимой
    # ценности рюкзака
    return max_value_matrix[len(items)][knapsack_capacity]
```

Задача 46

Логика решения этой задачи такова:

- Взять начальный и конечный IP-адреса диапазона адресов.
- Преобразовать начальный и конечный IP-адреса в их целочисленные представления, например "1.1.1.1" = 16843009. Это преобразование производится в два шага:
  - преобразовать каждую компоненту IP-адреса в двоичный формат, например "1.1.1.1" = b'\x01\x01\x01\x01';
    - найти подходящую функцию в библиотеке socket;
  - преобразовать двоичные данные в целое число, например b'\x01\x01\x01\x01' = 16843009;
    - найти подходящую функцию в библиотеке struct;
- Пробежаться в цикле от преобразованного начального IP-адреса до преобразованного конечного IP-адреса, для чего:
  - преобразовать каждое целое число в двоичный формат, например 16843009 = b'\x01\x01\x01\x01';
    - найти подходящую функцию в библиотеке struct;
  - преобразовать двоичные данные в IP-адрес, например b'\x01\x01\x01\x01' = "1.1.1.1";
    - найти подходящую функцию в библиотеке socket;
- Добавить получившиеся IP-адреса в список и вернуть результат.

Задача 47

Если вы никак не можете найти подход к этой задаче, то поинтересуйтесь, как работает алгоритм поиска в ширину. Для прохождения лабиринта вам придется реализовать его.

Ниже показана возможная логика решения:

```
def solve_maze(
    maze: list[list[int]],
    start_pos: tuple[int, int],
    end_pos: tuple[int, int]
) -> bool:

    # Инициализировать переменные
    num_rows, num_cols = len(maze), len(maze[0])
    queue = create_empty_queue()
    enqueue(queue, start_pos)
    mark_position_visited(maze, start_pos)

    # Цикл исследования лабиринта
    while queue не пуста:
        current_x, current_y = dequeue(queue)

        # Проверить, совпадает ли текущая позиция с конечной
        if current_x == end_pos.x and current_y == end_pos.y:
            return True

        # Исследовать соседние позиции
        for neighbor_x, neighbor_y in
            adjacent_position(current_x, current_y):

            if (в границах лабиринта и еще не посещалась):
                enqueue(queue, (neighbor_x, neighbor_y))
                mark_position_visited(maze,
                                      neighbor_x,
                                      neighbor_y)

    # Путь к конечной позиции не найден
    return False
```

Задача 48

Эту задачу можно решить с помощью двух шагов рекурсии и базового случая. Детали оставляю вам!

Задача 48.1 (дополнительная)

Первым делом нужно понять, как работает алгоритм обхода Морриса. Помимо объяснения в самой задаче, на эту тему есть немало ресурсов в сети. Только не подсматривайте в решения на Python!

Разобравшись в работе алгоритма, можете воспользоваться следующим псевдокодом:

```
def morris_traverse_inorder(root_node: TreeNode | None)
    -> Iterator[int]:

    current_node = root_node

    while current_node is not None:

        if current_node.left is None:
            yield значение current_node
            перейти к правому потомку current_node
        else:
            найти предшествующий узел predecessor_node, спустившись максимально
            далеко вправо от левого потомка current_node

            if правый потомок predecessor_node равен None:
                положить правый потомок predecessor_node равным
                current_node

                перейти к левому потомку current_node
            else:
                положить правый потомок predecessor_node равным None
                yield значение current_nod
                перейти к правому потомку current_node
```

Задача 49

При решении этой задачи следует использовать динамическое программирование, поэтому если вы не решили предыдущие задачи, например 45.1, рекомендую сделать это сейчас.

Нам надо продумать четыре основных момента.

1. Базовые случаи: если имеется 0, 1 или 2 ступеньки, то ответ уже известен, и мы можем сразу вернуть значение.
2. Инициализировать список динамического программирования нулями и уже известными значениями.
3. Заполнить список, пробежавшись в цикле от 3 до `total_stairs + 1`, вычисляя по ходу дела число комбинаций для каждого количества ступенек. Это можно сделать, используя ранее вычисленные значения в списке динамического программирования.
4. Вернуть последний элемент списка, равный полному числу способов подняться по лестнице.

Задача 49.1 (дополнительная)

Решение этой задачи похоже на решение задачи 49, только вместо того чтобы хранить число комбинаций, нужно хранить сами комбинации.

Основные изменения таковы:

- базовые случаи: вместо возврата 2, когда `total_stairs = 2`, мы возвращаем [[1, 1], [2]];
- список динамического программирования: вместо того чтобы сохранять 2 в позиции 2, мы сохраняем [[1, 1], [2]];
- рекурсивные шаги: вместо того чтобы складывать два предыдущих значения из списка динамического программирования, нужно как-то запоминать комбинации ступенек. Вопрос: как?

Задача 49.2 (дополнительная)

Это решение должно быть прямым обобщением решения задачи 49. Понадобится дополнительный базовый случай, два новых значения в начальном списке и небольшая модификация цикла.

Задача 50

Логика решения могла бы быть такой.

- Проверить, равна ли целевая сумма нулю. Если да, то никаких монет не нужно, просто возвращаем 0.
- Проверить, является ли список достоинств монет пустым. Если да, то с помощью имеющихся монет ничего разменять невозможно, поэтому возвращаем –1.
- Создать список `min_num_coins` длины `target_amount + 1` и инициализировать значением бесконечность все его элементы, кроме `min_num_coins[0]`, а в этот элемент записать 0. В этом списке мы будем хранить минимальное число монет, необходимых для размена любой возможной суммы от 0 до целевой.
- Теперь в цикле переберем все суммы от 1 до целевой суммы + 1:
  - для каждой суммы перебираем в цикле все имеющиеся достоинства монет:
    - проверить, можно ли использовать текущее достоинство для размена текущей суммы (т. е. что `current_target_amount - current_coin_value >= 0`);
    - если текущую монету можно использовать, то положить `min_num_coins[current_target_amount]` равным минимуму из ее достоинства и значения, полученного прибавлением 1 к `min_num_coins[current_target_amount - current_coin_value]`. На этом шаге вычисляется число монет, необходимых для размена текущей суммы.
- Наконец, возвращаем значение, хранящееся в `min_num_coins[target_amount]`, которое равно минимальному количеству монет заданных достоинств, необходимому для размена исходной целевой суммы.

Серьезные задачи: решения

Задача 1

```
def filter_strings_containing_a(input_strs: list[str])
     -> list[str]:
    # Использовать списковое включение с условием if, чтобы вернуть
    # новый список строк, содержащих букву "a"
    return [input_str for input_str in input_strs if "a" in input_str]
```

В этом решении используется списковое включение для создания нового списка строк, содержащих букву «a». Списковое включение обходит входной список строк и для каждой строки проверяет, содержит ли она букву «a». Если да, то строка включается в новый список. В конце новый список возвращается.

Для тех, кто незнаком со списковым включением, приведу эквивалентный код:

```
result = []

for input_str in input_strs:
    if "a" in input_str:
        result.append(input_str)

return result
```

Задача 2

```
def sum_if_less_than_fifty(num_one: int, num_two: int) -> int | None:
    # Вычислить сумму num_one и num_two, так мы избежим вычисления этой суммы
    # дважды в строке, где возвращается результат
    summed_value = num_one + num_two

    # Вернуть сумму, если она меньше 50, иначе вернуть None
    return summed_value if summed_value < 50 else None
```

В этом решении две строчки:

1) сложить два входных числа;
2) вернуть сумму, если она меньше 50.

Во второй строке используется выражение if (не путать с предложением if!). Выражение if является сокращенной записью следующего кода:

```
 if summed_value < 50:
    return summed_value
 else:
    return None
```

Задача 3

```
def sum_even(input_nums: list[int]) -> int:
    return sum(input_num
        for input_num
        in input_nums
        if input_num % 2 == 0)
```

Решение можно разбить на три шага.

1. Вызывается функция `sum` с генераторным выражением в качестве аргумента. Генераторное выражение обходит все элементы `input_num` входного списка `input_nums` и отфильтровывает нечетные числа, проверяя выполнение условия `input_num % 2 == 0` (т. е. делимости `input_num` на 2 без остатка).
2. Для каждого четного числа генераторное выражение отдает значение `input_num`.
3. Функция `sum` складывает все значения, отдаваемые генераторным выражением, и возвращает сумму всех четных чисел во входном списке.

Задача 4

```
def remove_vowels(input_str: str) -> str:
    vowels = "aeiouAEIOU"
    return ""
        .join(char for char in input_str if char not in vowels)
```

Сначала мы определяем строку, содержащую все строчные и заглавные гласные буквы. Затем используем генераторное выражение, чтобы обойти все буквы в строке `input_str` и отбросить гласные. В конце буквы, отданные генераторным выражением, объединяются в строку, которая и возвращается.

Задача 5

```
def get_longest_string(input_strs: list[str]) -> str:
    longest_str = ""
    for input_str in input_strs:
        if len(input_str) > len(longest_str):
            longest_str = input_str
    return longest_str
```

Мы заводим переменную `longest_str` и инициализируем ее пустой строкой. Затем обходим все строки во входном списке, и если длина текущей строки больше длины `longest_str`, то записываем в `longest_str` текущую строку. В конце функция возвращает `longest_str`.

Задача 6

```
def filter_even_length_strings(input_strs: list[str])
    -> list[str]:
    return [input_str for input_str in input_strs
        if len(input_str) % 2 == 0]
```

Здесь мы используем условное списковое включение, чтобы создать новый список, содержащий только строки с четным числом символов. Условие `len(input_str) % 2 == 0` означает «если остаток от деления длины строки на 2 равен 0, то включить строку в новый список».

Задача 7

```
def reverse_elements(input_nums: list[int]) -> list[int]:
    return input_nums[::-1]
```

Здесь мы используем для обращения списка срезы Python. Синтаксически срез записывается как пара квадратных скобок после списка, в которых указывается три числа: *начало:конец:шаг*.

Например, взяв список [1,2,3,4,5,6] и срез [0:4:2], мы получим результат [1, 3]. Действительно, определение среза говорит, что он начинается с индекса списка 0 (включительно), заканчивается индексом списка 4 (исключительно), а шаг равен 2.

В позиции с индексом 0 находится число 1. Затем мы сдвигаемся вперед на 2 позиции и берем элемент в позиции с индексом 2 – это число 3. Потом мы снова сдвигаемся вперед на 2 позиции, но этот индекс оказывается за конечной позицией среза, поэтому мы останавливаемся.

В этой задаче в качестве среза берется [::-1], т. е. ни начало, ни конец не определены, а шаг равен –1, что означает движение в противоположном направлении. Отсутствие начала и конца означает, что обрабатывается весь список, а поскольку шаг равен –1, обработка производится от его конца к началу.

Задача 8

```
def filter_type_str(input_list: list[str | int]) -> list[str]:
    return [list_item for list_item in input_list
        if isinstance(list_item, str)]
```

Здесь применяется условное списковое включение, как и в некоторых из предыдущих задач. Разница в том, что мы используем встроенную в Python функцию `isinstance`. Ей передается значение и тип, а она возвращает `True`, если значение принадлежит указанному типу, и `False` в противном случае.

Все элементы `list_item`, для которых `isinstance` возвращает `True`, добавляются в новый список, который затем возвращается.

Задача 9

```
def string_to_morse_code(input_str: str) -> str:
    morse_dict = {"a": ".-", "b": "-...", "c": "-.-.",
                  "d": "-..", "e": ".", "f": "..-.",
                  "g": "--.", "h": "....", "i": "..",
                  "j": ".---", "k": "-.-", "l": ".-..",
                  "m": "--", "n": "-.", "o": "---",
                  "p": ".--.", "q": "--.-", "r": ".-.",
                  "s": "...", "t": "-", "u": "..-",
                  "v": "...-", "w": ".--", "x": "-..-",
                  "y": "-.--", "z": "--..", "0": "-----",
                  "1": ".----", "2": "..---", "3": "...--",
                  "4": "....-", "5": ".....", "6": "-....",
                  "7": "--...", "8": "---..", "9": "----.",
                  ",": "--..--", ".": ".-.-.-", ":": "---...",
                  "?": "..--..", "'": ".----.", "-": "-....-",
                  "/": "-..-.", "(": "-.--.", ")": "-.--.-",
                  '"': ".-..-.", "@": ".--.-.", "=": "-...-",
                  "+": ".-.-.", "!": "-.-.--",
    }
    morse = []
    for char in input_str:
        if char != " ":
            char = char.lower()
            morse.append(morse_dict[char])
        else:
            morse.append("/")
    return " ".join(morse)
```

Здесь перебираются все символы строки `input_str`. Они преобразуются в нижний регистр, а затем в словаре `morse_dict` ищется код Морзе. Пробелы представляются косой чертой. Коды Морзе добавляются в список `morse`.

Когда все символы будут закодированы, функция возвращает список `morse`, вставив между соседними элементами пробелы.

Задача 10

```
def get_second_largest_number(input_nums: list[int])
    -> int | None:

    if len(input_nums) < 2:
        return None

    max_number = max(input_nums)
    input_nums.remove(max_number)
    second_max = max(input_nums)

    return second_max
```

Сначала проверяется длина списка `input_nums`. Если она меньше 2, т. е. в списке меньше двух чисел, то мы возвращаем `None`, потому что второго по величине числа просто нет.

Затем вычисляется второе по величине число, для чего мы находим и удаляем из списка максимальное число, а потом в оставшемся списке снова ищем максимальное число и возвращаем его.

Задача 11

```
def format_number_with_commas(input_num: int) -> str:
    return f"{input_num:,}"
```

Здесь мы используем для форматирования числа f-строку. Работает это так:

- буква f перед строкой означает, что это форматный строковый литерал;
- двоеточие внутри фигурных скобок означает, что мы хотим применить спецификатор формата;
- запятая после двоеточия означает, что мы хотим разделить запятыми группы по три разряда.

Задача 12

```
def string_to_ascii(input_str: str) -> list[int]:
    return [ord(char) for char in input_str]
```

Здесь используются встроенная в Python функция `ord` для преобразования каждой буквы строки в ее ASCII-код. Чтобы собрать результаты в список и вернуть его, применяется списковое включение.

Задача 12.1 (дополнительная)

```
def ascii_to_string(input_ascii_codes: list[int]) -> str:
    return "".join([chr(input_ascii_code)
        for input_ascii_code in input_ascii_codes])
```

Это решение похоже на предыдущее, но используется встроенная функция `chr` для преобразования числовых ASCII-кодов в соответствующие символы. Затем результаты объединяются в строку, которая и возвращается.

Задача 13

```
def has_vowel(input_str: str) -> bool:
    vowels = "aeiouAEIOU"
    for char in input_str:
        if char in vowels:
            return True
    return False

def filter_strings_with_vowels(input_strs: list[str])
    -> list[str]:

    return [input_str for input_str in input_strs
        if has_vowel(input_str)]
```

Здесь определена вспомогательная функция `has_vowel`, которая принимает строку и возвращает `True`, если она содержит гласную, и `False` в противном случае.

Функция `filter_strings_with_vowels` использует списковое включение, чтобы обойти список `input_strs` и оставить только строки, для которых `has_vowel` возвращает `True`.

Задача 14

```
def reverse_first_five_positions(input_nums: list[int])
    -> list[int]:

    return input_nums[:5][::-1] + input_nums[5:]
```

Здесь используются срезы и сложение списков для циклического сдвига первых пяти позиций. Понять, как это работает, проще всего на примере:

```
# Начальный вход
input_nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Отрезать первые пять чисел и обратить результат
input_nums[:5][::-1] = [5, 4, 3, 2, 1]

# Отрезать остальные числа после первых пяти
input_nums[5:] = [6, 7, 8, 9, 10]

# Сложить оба результата
input_nums[:5][::-1] + input_nums[:5] =
    [5, 4, 3, 2, 1, 6, 7, 8, 9, 10]
```

Задача 15

```
def filter_palindromes(input_strs: list[str]) -> list[str]:
    return [
        input_str
        for input_str in input_strs
        if input_str.lower() == input_str.lower()[::-1]
    ]
```

Здесь используется условное списковое включение с условием «если строка `input_str` в нижнем регистре равна обращенной строке `input_str` в нижнем регистре, то вернуть `True`».

Задача 16

```
def censor_python(input_strs: list[str]) -> list[str]:
    censored_chars = ["P", "Y", "T", "H", "O", "N"]
    return [
        "".join(
            ["X" if char.upper() in censored_chars else char
                for char in input_str]
        ) for input_str in input_strs
    ]
```

Решение состоит из нескольких шагов.

1. Инициализировать список `censored_chars` символами, которые следует вымарать: «P», «Y», «T», «H», «O», «N».
2. Выполнить списковое включение, которое обходит все строки (`input_str`) в списке `input_strs`.
3. Для каждой входной строки выполнить внутреннее списковое включение, которое обходит все символы (`char`) в строке.
4. Проверить, находится ли заглавный символ (`char.upper()`) в списке `censored_chars`.
5. Если символ найден в `censored_chars`, заменить его буквой `"X"`, иначе оставить исходный символ.
6. Собрать все символы в новую строку с помощью функции `"".join()`.
7. Собрать модифицированные строки в новый список.
8. Вернуть получившийся список.

Задача 17

```
def check_if_string_is_happy(input_str: str) -> bool:
    return not any(
        a == b or a == c or b == c
        for a, b, c in
            zip(input_str, input_str[1:], input_str[2:])
    )
```

Решение разбивается на следующие шаги.

1. Использовать функцию `zip()` для одновременного обхода троек соседних символов в строке `input_str`. Она соединяет каждый символ (`a`) со следующим за ним (`b`) и следующим за следующим (`c`).
2. Затем генераторное выражение внутри функции `any()` проверяет, есть ли среди этих символов равные.
3. Функция `any()` возвращает `True`, если хотя бы одно условие истинно, и `False` в противном случае.
4. Затем возвращается отрицание результата.

Задача 18

```
def get_number_of_digits(input_num: int) -> int:
    if input_num // 10 == 0:
        return 1
    return 1 + get_number_of_digits(input_num // 10)
```

В этом решении для вычисления результата используется рекурсия.

1. Первое предложение if возвращает 1, если `input_num`, поделенное на 10 (используется целочисленное деление), равно 0. Это условие истинно, если `input_num` – однозначное число.
2. Если это условие не выполнено, то выполняется шаг рекурсии. Функция `get_number_of_digits` вызывается снова с аргументом `input_num // 10`, т. е. из исходного числа удаляется последняя цифра.
3. Перед возвратом к результату рекурсивного вызова прибавляется 1, чтобы учесть текущую цифру в общем счетчике.
4. Рекурсия продолжается, пока не встретится базовый случай, после чего начинается раскрутка рекурсивных вызовов.

Задача 19

```
def get_tic_tac_toe_winner(input_board: list[list[str]])
     -> str | None:

    # Проверить строки
    for row in input_board:
        if row[0] == row[1] == row[2]:
            return row[0]

    # Проверить столбцы
    for i in range(3):
        if input_board[0][i] == input_board[1][i] == input_board[2][i]:
            return input_board[0][i]

    # Проверить диагонали
    if input_board[0][0] == input_board[1][1] == input_board[2][2]:
        return input_board[0][0]
    if input_board[0][2] == input_board[1][1] == input_board[2][0]:
        return input_board[0][2]

    return None
```

В решении производится четыре проверки.

1. Обходятся все три строки, и проверяется, нет ли среди них такой, в которой равны все три элемента.
2. Обходятся все три столбца, и проверяется, нет ли среди них такого, в котором равны все три элемента.
3. Проверяется диагональ, идущая из левого верхнего в правый нижний угол.
4. Проверяется диагональ, идущая из правого верхнего в левый нижний угол.

Если ни одно условие не выполнено, то игра завершилась вничью, и функция возвращает `None`.

Задача 20

```
def print_triangle(number_of_levels: int, symbol: str) -> None:
    for level in range(1, number_of_levels + 1):
        print(" "
            * (number_of_levels - level) + symbol
            * (2 * level - 1))
```

Здесь используется предложение `print` внутри цикла for для генерирования каждого уровня.

- Первая часть строки создается с помощью выражения `" " * (number_of_levels - level)`. Оно порождает строку пробелов, в которой число элементов равно разности между общим числом уровней (`number_of_levels`) и текущим уровнем (`level`). Таким образом создается отступ для каждого уровня: чем уровень выше, тем больше отступ.
- Вторая часть строки строится с помощью выражения `symbol * (2 * level - 1)`. Оно порождает строку заданных символов, в которой число элементов равно `(2 * level - 1)`. То есть мы вычисляем число символов на каждом уровне: на первом уровне 1 символ, на втором – 3, на третьем – 5 и т. д.

Задача 21

```
def fibonacci(sequence_number: int) -> int:
    if sequence_number in (0, 1):
        return sequence_number
    return fibonacci(sequence_number - 1)
        + fibonacci(sequence_number - 2)
```

В решении можно выделить две части.

1. Базовый случай: если `sequence_number` равен 0 или 1, то мы возвращаем это значение, потому что `fibonacci(0) = 0` и `fibonacci(1) = 1`.
2. Иначе функция дважды рекурсивно вызывает себя с аргументами `sequence_number - 1` и `sequence_number - 2`. Результаты рекурсивных вызовов складываются по достижении базового случая, когда начинается раскрутка стека.

Задача 22

```
def harmonic_sum(n: int) -> float:
    if n < 2:
        return 1
    return 1 / n + harmonic_sum(n - 1)
```

В этом решении выделяются три части.

1. Базовый случай: если *n* меньше 2, то мы возвращаем 1, потому что `harmonic_sum(0) = 1` и `harmonic_sum(1) = 1`.
2. Функция рекурсивно вызывает себя с аргументом `n - 1`. Этот рекурсивный вызов вычисляет гармоническую сумму для предыдущего значения `n`. Рекурсия продолжается, пока не будет достигнут базовый случай, и в этот момент начинается раскрутка стека и возврат значений.
3. После достижения базового случая функция начинает вычислять гармоническую сумму, прибавляя величину, обратную `n`, к результату рекурсивного вызова. Когда стек вызовов опустеет, функция вернет окончательную гармоническую сумму.

Задача 23

```
def xor(input_a: str, input_b: str) -> str:
    result = ""
    for i in range(min(len(input_a), len(input_b))):
        if input_a[i] == input_b[i]:
            result += "0"
        else:
            result += "1"
    return result
```

Здесь в цикле for перебираются символы строк `input_a` и `input_b`. Для каждой пары символов проверяется, одинаковы они или нет. Если одинаковы, то в конец результирующей строки дописывается 0, иначе 1.

Задача 24

```
from typing import Any

def my_zip(input_list_a: list[Any], input_list_b: list[Any])
    -> list[tuple[Any, Any]]:

    zipped_result = []
    for i in range(min(len(input_list_a), len(input_list_b))):
        zipped_result.append((input_list_a[i], input_list_b[i])
    )
    return zipped_result
```

Здесь сначала инициализируется пустой список `zipped_result`, в котором будут храниться результирующие кортежи.

Затем мы используем цикл for по всем индексам `i` самого короткого списка. Тем самым гарантируется, что не возникнет ошибка `IndexError`, если списки имеют разную длину.

В цикле функция добавляет кортеж в конец списка `zipped_result`. Кортеж содержит пару элементов с индексом `i`, взятых из списков `input_list_a` и `input_list_b`.

По завершении цикла возвращается результат.

Задача 25

```
def is_valid_equation(input_equation: str) -> bool:
    try:
        left_num, operator, right_num, _, result_num
            = input_equation.split()
        if operator == "+":
            return int(left_num) + int(right_num) == int(result_num)
        if operator == "-":
            return int(left_num) - int(right_num) == int(result_num)
        return False
    except ValueError:
        return False
```

Здесь мы сначала разбиваем строку `input_equation` на части по пробелам. Каждая часть загружается в соответствующие переменные, причем знак равенства загружен в переменную с именем `"_"`, потому что его значение нас не интересует.

После этого выполняются два предложения if: одно – для случая, когда оператором является плюс, а другое – для случая, когда это минус. Если условие в каком-то из предложений if равно `True`, то мы вычисляем выражение и проверяем, равны ли его части.

Если оператор – не плюс и не минус, то мы возвращаем `False`. Если при выполнении кода в блоке try возникнет исключение `ValueError`, значит, равенство не удалось правильно разделить на ожидаемые компоненты. В этом случае выполняется блок except, и функция возвращает `False`.

Задача 26

```
from typing import Any

def rotate_list_left(input_list: list[Any], rotate_amount: int)
    -> list[Any]:

    rotate_amount %= len(input_list)

    return input_list[rotate_amount:]
        + input_list[:rotate_amount]
```

Здесь возвращается результат конкатенации двух срезов списка `input_list` с помощью оператора `+`.

1. Сначала мы используем оператор деления по модулю, чтобы уменьшить величину сдвига `rotate_amount`, сделав ее меньше размера списка. Действительно, если у нас есть список длины 5 и мы циклически сдвинем его 5 раз, то в результате получим тот же самый список. Так что если длина списка равна 5, а `rotate_amount` равно 7, то результат будет такой же, как если бы `rotate_amount` было равно 2.
2. Теперь перейдем к срезам. Первый срез, `input_list[rotate_amount:]`, дает элементы `input_list` с индексами от `rotate_amount` до конца списка. Эти элементы должны быть перемещены в начало списка в процессе циклического сдвига.
3. Второй срез, `input_list[:rotate_amount]`, дает элементы `input_list` от начала списка до индекса `rotate_amount`. Эти элементы должны быть перемещены в конец списка в процессе сдвига.

Путем конкатенации обоих срезов функция циклически сдвигает элементы `input_list` на `rotate_amount` влево.

Задача 27

```
def find_adjacent_nodes(
    adj_matrix: list[list[int]],
    start_node: int
) -> list[int]:

    return [i for i, is_connected in
        enumerate(adj_matrix[start_node]) if is_connected]
```

Вас, наверное, удивило, что решение этой задачи занимает всего одну строчку, но в этой строчке много чего происходит! Разберемся.

1. Мы используем списковое включение, чтобы обойти все элементы списка `adj_matrix[start_node]`. Этот подсписок представляет строку матрицы смежности, соответствующую `start_node`.
2. Внутри спискового включения каждый элемент представлен переменными `i` и `is_connected`. В переменной `i` хранится индекс текущего элемента, а `is_connected` содержит значение true (1) или false (0), показывающее, связан ли один узел с другим.
3. Условие в списковом включении, `is_connected`, отфильтровывает элементы, не связанные со `start_node`. Это условие истинно, только если `is_connected` не равно 0.
4. Отфильтрованные элементы собираются в новый список, созданный списковым включением, так что в результате создается список индексов, представляющих смежные узлы, связанные со `start_node`.
5. Наконец, список смежных узлов возвращается в качестве результата функции.

Задача 28

```
def count_peaks_valleys(price_action: list[int])
    -> tuple[int, int]:

    peaks = 0
    valleys = 0
    for price_idx in range(1, len(price_action) - 1):
        if (
            price_action[price_idx - 1]
            < price_action[price_idx]
            > price_action[price_idx + 1]
        ):
            peaks += 1
        elif (
            price_action[price_idx - 1]
            > price_action[price_idx]
            < price_action[price_idx + 1]
        ):
            valleys += 1
    return peaks, valleys
```

Здесь список `price_action` обходится, начиная со второго и до предпоследнего элемента. Внутри цикла проверяются два условия.

1. Для распознавания пика должны быть истинны следующие условия:
   a) цена перед текущей должна быть меньше текущей;
   b) цена после текущей должна быть больше текущей.
2. Для распознавания впадины должны быть истинны следующие условия:
   a) цена перед текущей должна быть больше текущей;
   b) цена после текущей должна быть меньше текущей.

Если обнаружен пик или впадина, то мы увеличиваем счетчики `peaks` или `valleys` соответственно, а по выходе из цикла возвращаем эти счетчики.

Задача 29

```
tap_code_map = {
    "a": ". .",       "b": ". ..",       "c": ". ...",
    "d": ". ....",    "e": ". .....",    "f": ".. .",
    "g": ".. ..",     "h": ".. ...",     "i": ".. ....",
    "j": ".. .....",  "l": "... .",      "m": "... ..",
    "n": "... ...",   "o": "... ....",   "p": "... .....",
    "q": ".... .",    "r": ".... ..",    "s": ".... ...",
    "t": ".... ....", "u": ".... .....", "v": "..... .",
    "w": "..... ..",  "x": "..... ...",  "y": "..... ....",
    "z": "..... .....",
}

def tap_code_to_english(input_code: str) -> str:
    tap_code_map_inv = {v: k for k, v in tap_code_map.items()}

    if len(input_code) == 0:
        return ""

    # Разбить код перестукивания на слова, преобразовать каждое слово, а затем
    # конкатенировать результаты.

    words_as_tap_code = input_code.split("   ")
    return " ".join(
        [
            "".join(
                tap_code_map_inv[letter_as_tap_code]
                for letter_as_tap_code
                    in word_as_tap_code.split("  ")
            )
            for word_as_tap_code in words_as_tap_code
        ]
    )
```

Сначала мы изменяем отображение `tap_code_map`, так чтобы значения стали ключами, а ключи – значениями. Можно было бы с самого начала определить отображение именно так, но рассматривать буквы как ключи, а их коды как значения более естественно.

Затем идет простая проверка – мы возвращаем пустую строку, если длина `input_code` равна 0.

Далее начинается основная часть обработки, где мы сначала разбиваем входной код на список закодированных слов (напомним, что слова в коде перестукивания разделяются тремя пробелами). Затем список закодированных слов обрабатывается следующим образом.

1. Внутри спискового включения для каждого слова:
   a) разбить слово на список закодированных букв (напомним, что буквы в коде перестукивания разделяются двумя пробелами);
   b) найти код в отображении `tap_code_map_inv` и взять соответствующую ему букву;
   c) объединить полученные в результате преобразования буквы.
2. С помощью функции `join` преобразовать возвращенный списковым включением список в строку, вставив между словами пробелы.

Задача 30

```
def find_zero_sum_triplets(input_nums: list[int])
    -> list[tuple[int, int, int]]:

    zero_sum_triplets = []

    for i in range(len(input_nums)):
        for j in range(i + 1, len(input_nums)):
            for k in range(j + 1, len(input_nums)):
                if input_nums[i] + input_nums[j]
                    + input_nums[k] == 0:
                    zero_sum_triplets.append((i, j, k))

    return zero_sum_triplets
```

Сначала мы инициализируем пустой список `zero_sum_triplets`, в котором будут храниться найденные тройки. Для этого в трех вложенных циклах for мы обходим все возможные комбинации трех элементов из списка `input_nums`:

1. Во внешнем цикле по переменной `i` перебираются индексы `input_nums` от 0 до длины списка.
2. В среднем цикле по переменной `j` перебираются индексы от `i + 1` до длины списка. Поэтому `j` всегда будет больше `i`, что предотвращает появление повторяющихся комбинаций.
3. Во внутреннем цикле по переменной `k` перебираются индексы от `j + 1` до длины списка. Поэтому `k` всегда будет больше `i` и `j`, что опять-таки предотвращает появление повторяющихся комбинаций.

Во внутреннем цикле проверяется, равна ли 0 сумма элементов списка `input_nums` с индексами `i`, `j`, `k`. Если да, то мы нашли тройку, дающую в сумме 0. В таком случае кортеж `(i, j, k)`, представляющий индексы трех элементов, добавляется в конец списка `zero_sum_triplets`.

После всех итераций функция возвращает список `zero_sum_triplets`, содержащий найденные тройки.

Задача 31

```
from typing import Any

def param_count(*args: Any) -> int:
    return len(args)
```

Здесь определяется функция `param_count`, которая принимает переменное число аргументов, используя синтаксис `*args`. Функция возвращает длину кортежа `args`, пользуясь функцией `len`.

Задача 31.1 (дополнительная)

Есть два подхода к решению этой задачи. Первое решение, вероятно, будет ближе к написанному вами, потому что это первое, что приходит на ум. Во втором решении используются итерируемые объекты, и оно ближе к тому, как в Python на самом деле реализована функция `zip`.

```
from typing import Any

def my_zip_one(*input_lists: list[Any]) -> list[tuple[Any, ...]]:
    if len(input_lists) == 0:
        return []

    shortest_length = min(len(input_list) for input_list
        in input_lists)
    zipped_result = [tuple() for _ in range(shortest_length)]

    for i in range(shortest_length):
        for input_list in input_lists:
            zipped_result[i] += (input_list[i],)

    return zipped_result
```

Это решение можно разбить на пять частей.

1. Если список `input_lists` пуст, вернуть пустой список.
2. Вычислить длину самого короткого списка в `input_lists`, воспользовавшись функциями `min` и `len` в сочетании с генераторным выражением.
3. Создать список пустых кортежей, в котором число кортежей равно длине самого короткого из входных списков.
4. Создать внешний цикл по `i` от 0 до длины самого короткого списка.
   a. Внутри этого цикла обойти в цикле все списки в `input_lists`, добавляя `i`-й элемент в `i`-й результирующий кортеж.
   b. Здесь мы пользуемся сложением кортежей, например:
      i) если `zipped_result[i] = (1, 2, 3)`
      ii) и выполняется предложение `zipped_result[i] += (4,)`
      iii) то в результате получится кортеж (1, 2, 3, 4).
5. Напоследок мы возвращаем результат.

Задача 31.1 – другое решение

```
from typing import Any
from collections.abc import Iterator

def my_zip_two(*input_lists: list[Any]) -> Iterator[tuple[Any, ...]]:
    sentinel = object()
    input_lists_iterators = [iter(input_list) for input_list in input_lists]
    while input_lists_iterators:
        zipped_result = []
        for input_list_iterator in input_lists_iterators:
            elem = next(input_list_iterator, sentinel)
            if elem is sentinel:
                return
            zipped_result.append(elem)
        yield tuple(zipped_result)
```

Это решение ближе к реализации функции `zip` в Python. Вот что здесь происходит.

1. Создается объект `sentinel`. Он будет нужен для обнаружения момента, когда входной список исчерпан.
2. Для каждого входного списка в `input_lists` с помощью функции `iter()` создается итератор, который позволяет обходить элементы своего списка.
3. Функция входит в цикл `while`, который продолжается, пока все входные списки не будут исчерпаны.
   a. В цикле создается новый пустой список `zipped_result`.
   b. Для каждого итератора по входному списку в `input_lists_iterators` функция пытается получить следующий элемент, вызывая `next(input_list_iterator, sentinel)`.
      i. Если следующий элемент существует, то он добавляется в конец списка `zipped_result`.
      ii. Если итератор исчерпан (т. е. `next()` вернула объект `sentinel`), то функция возвращает управление досрочно с помощью предложения return.
   c. После обхода всех итераторов входных списков из элементов в списке `zipped_result` создается кортеж с помощью функции `tuple()`.
   d. Функция отдает кортеж в предложении yield, что превращает ее в генератор кортежей. Это означает, что функцию можно обходить, получая по одному кортежу за раз.
4. Если все итераторы входных списков исчерпаны, то функция завершается и больше кортежей не генерируется.

Задача 32

```
def contains_python_chars(input_str: str) -> bool:
    input_str = input_str.lower()
    python_chars = set("python")

    for i in range(len(input_str) - len(python_chars) + 1):
        if set(input_str[i:i + len(python_chars)]) == python_chars:
            return True
    return False
```

Здесь мы сначала преобразовываем входную строку в нижний регистр с помощью функции `lower`, чтобы функция не зависела от регистра. Затем проверяется, является ли строка перестановкой слова python.

1. Из букв слова `"python"` создается множество `python_chars`. В дальнейшем мы будем использовать его для сравнения.
2. Функция входит в цикл for от 0 до длины строки `input_str` минус длина `python_chars` плюс 1. Это гарантирует, что мы не выйдем за границы строки, когда будем вычислять срезы `input_str` в цикле for.
   a. Внутри цикла из `input_str` вырезается подстрока, начиная с позиции `i` длиной, равной длине `python_chars`: `input_str[i:i + len(python_chars)]`.
   b. Вырезанная подстрока преобразуется в множество функцией `set()`, а затем сравнивается с `python_chars` с помощью оператора `==`.
   c. Если оба множества равны, значит, подстрока содержит все буквы слова `"python"`, и функция возвращает `True`.
   d. Если подстрока не совпадает с множеством `python_chars`, то цикл продолжается, пока не будут проверены все подстроки.
3. Если цикл завершился, а совпадений не найдено, то функция возвращает `False`, показывая, что `input_str` не содержит никакой анаграммы слова `"python"`.

Задача 33

```
def is_prime(input_num: int) -> bool:
    if input_num < 2:
        return False

    for i in range(2, input_num):
        if input_num % i == 0:
            return False
    return True

def find_primes(input_nums: list[int]) -> list[int]:
    return [input_num for input_num in input_nums
        if is_prime(input_num)]
```

Здесь определена вспомогательная функция `is_prime`, которая принимает целое число и возвращает `True`, если это число простое, а иначе `False`. Функция `find_primes` пользуется списковым включением и функцией `is_prime` для создания нового списка, который содержит только простые числа из входного списка.

Задача 34

```
def rot13(input_str: str) -> str:
    result = ""

    for char in input_str:
        if char.isalpha():
            a_code = ord("A") if char.isupper() else ord("a")
            char = chr((ord(char) - a_code + 13) % 26 + a_code)

        result += char

    return result
```

В самом начале мы инициализируем пустую строку `result`, а затем организуем цикл по строке `input_str`:

1. В цикле мы проверяем, является ли `char` буквой, пользуясь методом `isalpha`. Если да, то функция переходит к преобразованию ROT13.
2. В переменную `a_code` записывается ASCII-код заглавной буквы `"A"` или строчной буквы `"a"` в зависимости от регистра текущего символа.
3. Теперь к символу применяется преобразование ROT13, которое мы рассмотрим более подробно:
   a) `ord(char)`: функция `ord()` возвращает ASCII-код символа `char`;
   b) `(ord(char) - a_code + 13)`: из ASCII-кода символа вычитается значение `a_code`. Этот шаг призван нормализовать значение символа относительно начала алфавита (его строчной или заглавной части);
   c) `% 26`: берется остаток от деления на 26 значения, полученного на предыдущем шаге. Эта операция приводит результат к диапазону от 0 до 25 – на случай, если результат вычисления оказался вне этого диапазона;
   d) `+ a_code`: получив остаток от деления на 26, мы прибавляем к нему `a_code`. Этот шаг приводит результат к диапазону ASCII-кодов, соответствующему строчным и заглавным буквам;
   e) `chr(...)`: и на последнем шаге полученный ASCII-код преобразуется в символ с помощью функции `chr()`. Это дает нам символ, зашифрованный шифром ROT13.

4. Цикл продолжается, пока не будут обработаны все символы `input_str`, после чего возвращается результат.

Пример

Наименее понятная часть решения – строчка `char = chr((ord(char) - a_code + 13) % 26 + a_code)`, поэтому разберем ее на примере:

```
char = "Y"
a_code = ord("A") = 65

chr(ord(char)) – a_code + 13) % 26 + a_code)
 = chr((89 – 65 + 13) % 26 + a_code)   # вычислены ord(char)) и a_code
 = chr(37 % 26 + a_code)               # вычислено (89 – 65 + 13) = 37
 = chr(11 + a_code)                    # вычислено 37 % 26 = 11
 = chr(11 + 65)                        # вычислено a_code
 = chr(76)                             # вычислено 11 + 65 = 76
 = "L"
```

Задача 35

```
def get_parentheses_groups(input_str: str) -> list[str]:
    open_indices = []
    groups = []
    start_index = 0

    for i, char in enumerate(input_str):
        if char == "(":
            open_indices.append(i)
        elif char == ")":
            if open_indices:
                open_indices.pop()
                if not open_indices:
                    group = input_str[start_index:i + 1]
                        .replace(" ", "")
                    groups.append(group)
                    start_index = i + 1

    return groups
```

1. Сначала инициализируются три переменные.
   a. Пустой список `open_indices`, в котором будут храниться индексы встретившихся открывающих скобок.
   b. Пустой список `groups`, в котором будут храниться выделенные группы скобок.
   c. Переменная `start_index`, инициализированная значением 0. Она представляет начальный индекс текущей группы.
2. Функция входит в цикл, в котором перебирает все символы строки `input_str`, запоминая также соответствующий индекс `i`. В теле цикла проверяются два условия.
   a. Если `char` равен `"("`:
      i) индекс `i` добавляется в конец списка `open_indices`.
   b. Если `char` равен `")"`:
      i) если имеются незакрытые открывающие скобки (список `open_indices` не пуст), то индекс последней открывающей скобки удаляется из `open_indices` методом `open_indices.pop()`. Это означает, что найдена парная закрывающая скобка;
      ii) если после удаления индекса открывающей скобки список `open_indices` оказался пуст, значит, найдена полная группа скобок. В таком случае мы:

1) вырезаем группу скобок из входной строки и удаляем все пробелы между ними;
2) добавляем найденную группу скобок в список `groups`;
3) записываем в `start_index` значение `i + 1`, равное индексу начала следующей группы.

3. По завершении цикла возвращается результирующий список групп.

Задача 36

```
def matrix_multiply(
    left_matrix: list[list[int]], right_matrix: list[list[int]]
) -> list[list[int]]:

    num_left_rows = len(left_matrix)
    num_left_cols = len(left_matrix[0])
    num_right_cols = len(right_matrix[0])

    if num_left_rows != num_right_cols:
        return None

    result_matrix = [[0 for _ in range(num_right_cols)]
        for _ in range(num_left_rows)]

    for i in range(num_left_rows):
        for k in range(num_left_cols):
            for j in range(num_right_cols):
                result_matrix[i][j]
                    += left_matrix[i][k] * right_matrix[k][j]

    return result_matrix
```

Поясним, как устроено это решение.

1. Сначала функция определяет размерности левой и правой матриц.
   a. `num_left_rows` устанавливается равным числу строк в левой матрице, полученному от функции `len(left_matrix)`.
   b. `num_left_cols` устанавливается равным числу столбцов в левой матрице, полученному от функции `len(left_matrix[0])`.
   c. `num_right_cols` устанавливается равным числу столбцов в правой матрице, полученному от функции `len(right_matrix[0])`.
2. Если перемножить матрицы невозможно, потому что `num_left_rows != num_right_cols`, то возвращается `None`.
3. Результирующая матрица `result_matrix` инициализируется вложенным списком нулей с помощью спискового включения.
   a. Размерность `result_matrix` равна `num_left_rows` x `num_right_cols`.

4. Далее следуют три вложенных цикла для умножения матриц:
   a) во внешнем цикле по переменной `i` перебираются строки левой матрицы;
   b) в среднем цикле по переменной `k` перебираются столбцы левой матрицы и строки правой матрицы;
   c) во внутреннем цикле по переменной `j` перебираются столбцы правой матрицы и вычисляются элементы результирующей матрицы.
      i. К элементу результирующей матрицы в позиции `[i][j]` прибавляется произведение соответственных элементов из левой и правой матриц.
5. По завершении всех итераций возвращается результирующая матрица, содержащая произведение входных.

Задача 37

```
def gcd(num_one: int, num_two: int) -> int:
    if num_two == 0:
        return num_one
    return gcd(num_two, num_one % num_two)
```

Сначала определяется базовый случай рекурсии: `if num_two == 0`. Если `num_two` равно 0, то `num_one` и есть НОД двух чисел, потому что 0 делится на любое число. В таком случае функция сразу возвращает `num_one`.

В противном случае функция рекурсивно вызывает себя.

1. Функция вызывается с аргументами `num_two` и `num_one % num_two`. Таким образом, мы рекурсивно сводим задачу вычисления НОД к меньшей.
2. В этом вызове предыдущее `num_two` становится новым `num_one`, а остаток от деления `num_one` на `num_two` становится новым `num_two`.
3. Рекурсия продолжается, пока `num_two` не станет равным 0 (базовый случай).

По достижении базового случая функция возвращает НОД – значение переменной `num_one`.

Пример

```
gcd(10, 15)

gcd(15, 10 % 15)
 = gcd(15, 10)    # 10 % 15 = 10

gcd(10, 15 % 10)
 = gcd(10, 5)     # 15 % 10 = 5

gcd(5, 10 % 5)
 = gcd(5, 0)     # Базовый случай, т. к. num_two == 0
 = 5
```

Задача 38

```
def find_pairs_summing_to_target(
    input_nums: list[int], target: int
) -> list[tuple[int, int]]:

    pairs = []

    for left_idx, left_num in enumerate(input_nums):
        for _, right_num in enumerate(
            input_nums[left_idx + 1:]
        ):

            if left_num + right_num == target:
                if (left_num, right_num) not in pairs:
                    pairs.append((left_num, right_num))

    return pairs
```

Сначала инициализируется пустой список `pairs`. После этого начинается вложенный цикл for:

1. Во внешнем цикле обходится список `input_nums`, при этом применяется функция `enumerate`, чтобы получить сразу индекс `left_idx` и соответствующее ему значение `left_num`.
   a. Во внутреннем цикле обходится срез списка `input_nums` от `left_idx + 1` до конца. Тем самым гарантируется отсутствие повторяющихся пар.
   b. Во внутреннем цикле сумма `left_num` и `right_num` сравнивается со значением `target`. Если они равны, то кортеж `(left_num, right_num)` добавляется в список `pairs`.

По завершении обоих циклов функция возвращает список `pairs`, содержащий все пары чисел, в сумме равных `target`.

Задача 38.1 (дополнительная)

```
def find_pairs_summing_to_target_bonus(
    input_nums: list[int], target: int
) -> list[tuple[int, int]]:

    pairs = set()
    seen = set()

    for input_num in input_nums:

        if target - input_num in seen:
            pairs.add((input_num, target - input_num))
        else:
            seen.add(input_num)

    return list(pairs)
```

В этом решении используется один цикл for и множество с целью добиться временной сложности $O(n)$, что эффективнее вложенных циклов в предыдущем решении.

1. Сначала инициализируются две переменные.
   a. Пустое множество `pairs` для хранения пар чисел, в сумме дающих `target`.
   b. Пустое множество `seen` для запоминания чисел, которые уже встречались при обходе `input_nums`.
2. Затем начинается цикл обхода списка `input_nums`.
   a. Для каждого элемента `input_num` функция проверяет, присутствует ли разность между `target` и `input_num` в множестве `seen`. Если да, значит, мы нашли пару чисел, дающих в сумме `target`. В таком случае добавляем кортеж (`input_num, target - input_num`) в множество `pairs`.
   b. Если же разности нет в множестве `seen`, значит, для текущего элемента `input_num` нет парного, который в сумме с ним давал бы `target`. В таком случае мы добавляем `input_num` в множество `seen` для использования в будущем.
3. Цикл продолжается, пока не будут обработаны все элементы списка `input_nums`, после чего множество `pairs` преобразуется в список и возвращается.

Задача 39

```
def tower_of_hanoi(num_disks: int, source: str, aux: str, target: str)
    -> None:

    if num_disks > 0:
        tower_of_hanoi(num_disks - 1, source, target, aux)
        print(f"Переместить диск {num_disks} с {source} на {target}")
        tower_of_hanoi(num_disks - 1, aux, source, target)
```

Здесь функция рекурсивно вызывает себя дважды. Объясним, как это работает.

1. Базовый случай рекурсии: если `num_disks` равно 0 или отрицательно, функция возвращает управление.
2. В противном случае функция рекурсивно вызывает себя с числом дисков `num_disks - 1`. Но при этом роли аргументов `aux` и `target` меняются местами.
3. После этого рекурсивного вызова печатается сообщение `print(f"Переместить диск {num_disks} с {source} на {target}")`.
4. Далее производится второй рекурсивный вызов с числом дисков `num_disks - 1`, но на этот раз меняются роли аргументов `source` и `aux`. То есть мы перемещаем оставшиеся диски со стержня `aux` на стержень `target`, используя стержень `source` как вспомогательный.

Выполняя эти шаги, алгоритм рекурсивно перемещает диски с исходного стержня на целевой, используя вспомогательный стержень как временное хранилище, гарантируя при этом, что больший диск никогда не будет положен на меньший.

Задача 40

```
def insertion_sort(input_nums: list[int]) -> list[int]:
    for key_index in range(1, len(input_nums)):
        value_to_insert = input_nums[key_index]
        previous_index = key_index - 1

        while previous_index >= 0
            and value_to_insert < input_nums[previous_index]:

            input_nums[previous_index + 1] =
                input_nums[previous_index]

            previous_index -= 1

        input_nums[previous_index + 1] = value_to_insert

    return input_nums
```

Здесь мы используем для сортировки цикл for.

1. В цикле for обходится диапазон от 1 до длины списка `input_nums`. Это значит, что мы начинаем со второго элемента списка.
2. Внутри цикла значение элемента с текущим индексом `key_index` сохраняется в переменной `value_to_insert`.
3. Переменной `previous_index` присваивается значение `key_index - 1`, т. е. индекс предыдущего элемента.
4. В цикле while `value_to_insert` сравнивается с предшествующими элементами (начиная с `previous_index`), и те из них, которые больше `value_to_insert`, сдвигаются вправо. Цикл продолжается, пока `previous_index` больше или равен 0 и элемент с индексом `previous_index` больше `value_to_insert`.
   a. Внутри цикла while элемент с индексом `previous_index` сдвигается на одну позицию вправо (`input_nums[previous_index + 1] = input_nums[previous_index]`). Так мы освобождаем место для вставки `value_to_insert` в нужную позицию.
   b. `previous_index` уменьшается на 1, чтобы перейти к следующему элементу, более близкому к началу списка.
5. По завершении цикла while найдена правильная позиция для `value_to_insert`, и это значение записывается в `input_nums[previous_index + 1]`.
6. Производится переход к следующему элементу – и так до тех пор, пока все элементы не будут обработаны.

Наконец, отсортированный список `input_nums` возвращается в качестве результата функции.

Задача 41

```
roman_map = { 1000: "M", 900: "CM", 500: "D", 400: "CD",
    100: "C", 90: "XC", 50: "L", 40: "XL", 10: "X",  9: "IX",
    5: "V", 4: "IV", 1: "I",
}

def int_to_roman(input_num: int) -> str:
    result = ""

    for roman_int_value in sorted(
        roman_map.keys(), reverse=True
    ):
        while input_num >= roman_int_value:
            result += roman_map[roman_int_value]
            input_num -= roman_int_value

    return result
```

Сначала мы инициализируем пустую строку `result`, в которой будет храниться преобразованное значение.

Затем в цикле for перебираются ключи словаря `roman_map` в порядке убывания. Поэтому наибольшие возможные римские цифры рассматриваются первыми.

Внутри цикла for имеется цикл while, в котором:

1) `roman_map[roman_int_value]` дописывается в конец результата с помощью конкатенации строк (`result += roman_map[roman_int_value]`);
2) из текущего значения `input_num` вычитается значение `roman_int_value`, чтобы уменьшить преобразуемое число.

По завершении цикла while начинается следующая итерация цикла for с уменьшенным значением `roman_int_value`.

По завершении цикла for возвращается результат.

Задача 41.1 (дополнительная)

```
roman_map = { 1000: "M", 900: "CM", 500: "D", 400: "CD",
    100: "C", 90: "XC", 50: "L", 40: "XL", 10: "X",  9: "IX",
    5: "V", 4: "IV", 1: "I",
}

def roman_to_int(input_str: str) -> int:
    roman_map_swapped = {v: k for k, v in roman_map.items()}

    result, prev_value = 0, 0

    for char in input_str:
        value = roman_map_swapped[char]

        if value > prev_value:
            result -= prev_value
            result += value - prev_value
        else:
            result += value

        prev_value = value

    return result
```

Сначала определим функцию `roman_to_int`. Она работает следующим образом.

1. Путем перестановки ключей и значений словаря `roman_map` создается новый словарь `roman_map_swapped`. Это нужно, чтобы мы могли использовать в качестве ключей значения римских цифр.
2. Переменной `result` присваивается значение 0. В ней будет храниться результат.
3. Переменной `prev_value` присваивается значение 0. В этой переменной хранится предыдущее целое значение, встретившееся в процессе итераций.
4. В цикле for обходим все символы строки `input_str` и выполняем следующие действия:
   a) получить целое значение текущей римской цифры, поискав ее в словаре `roman_map_swapped`;
   b) сравнить значения `value` и `prev_value`. Благодаря этому сравнению мы узнаем, какую операцию производить при обработке текущей римской цифры: сложение или вычитание.

i. Если value больше prev_value, то вычитаем prev_value из result и прибавляем к result разность между value и prev_value. Это случай субтрактивной комбинации римских цифр (например, IV = 4, CM = 900).
ii. Если value не больше prev_value, то мы прибавляем значение прямо к result. Это случай аддитивной комбинации римских цифр (например, VI = 6, XI = 11);

c) присвоить значение value переменной prev_value перед следующей итерацией.

По выходе из цикла функция возвращает переменную result.

```
def int_roman_converter(to_convert: str | int) -> int | str:
    # римские числа в int
    if isinstance(to_convert, str):
        return roman_to_int(to_convert)

    # int в римские числа
    if isinstance(to_convert, int):
        return int_to_roman(to_convert)

    return None
```

Функция int_roman_converter – последний кусочек пазла. Она написана в предположении, что если to_convert имеет тип str, то мы хотим преобразовать римское число в int, а если to_convert имеет тип int, то мы хотим преобразовать int в римское число.

Если тип to_convert не str и не int, то мы просто возвращаем None.

Задача 42

```
def bitwise_add(num_one: int, num_two: int) -> int:
    while num_two != 0:
        carry = num_one & num_two
        num_one = num_one ^ num_two
        num_two = carry << 1

    return num_one
```

Здесь цикл while используется для обработки переносов в процессе сложения. В цикле производятся следующие действия.

1. Вычисляется перенос путем выполнения операции поразрядного И (`&`) между `num_one` и `num_two`, результат присваивается переменной `carry`.
2. В `num_one` записывается результат поразрядной операции ИСКЛЮЧАЮЩЕЕ ИЛИ (`^`) между `num_one` и `num_two`. На этом шаге два числа складываются без учета переноса.
3. В `num_two` записывается результат сдвига `carry` на один разряд влево (`carry << 1`). Сдвиг `carry` влево на один разряд подготавливает его к прибавлению на следующей итерации.

Когда `num_two` обращается в нуль, происходит выход из цикла, и окончательный результат – сумма двух входных чисел – оказывается в `num_one`. Эту величину функция и возвращает.

Продолжить объяснение лучше на примере.

```
num_one = 9 (двоичное 1001)
num_two = 5 (двоичное 0101)

Начинается цикл while

Итерация 1:
carry = num_one & num_two = 1001 & 0101 = 0001 (carry: 1)
num_one = num_one ^ num_two = 1001 ^ 0101 = 1100 (num_one: 12)
num_two = carry << 1 = 0001 << 1 = 0010 (num_two: 2)

Итерация 2:
carry = num_one & num_two = 1100 & 0011 = 0000 (carry: 0)
num_one = num_one ^ num_two = 1100 ^ 0010 = 1110 (num_one: 14)
num_two = carry << 1 = 0000 << 1 = 0000 (num_two: 0)

Выход из цикла while, потому что num_two стало равно 0. Конечный результат
находится в num_one и равен 14.
```

Задача 43

```
def binary_search(sorted_list: list[int], value_to_find: int)
    -> int:

    low, mid = 0, 0
    high = len(sorted_list) - 1

    while low <= high:
        mid = (high + low) // 2

        # Если value_to_find больше, игнорировать левую половину
        if value_to_find > sorted_list[mid]:
            low = mid + 1

        # Если value_to_find меньше, игнорировать правую половину
        elif value_to_find < sorted_list[mid]:
            high = mid - 1

        # value_to_find совпадает с mid
        else:
            return mid

    # Если мы оказались здесь, то искомого элемента нет в списке
    return -1
```

Есть два способа реализовать двоичный поиск: рекурсивный и итеративный. У обоих решений временная сложность $O(\log n)$, но у итеративного способа пространственная сложность $O(1)$, а у рекурсивного – $O(\log n)$.

В показанном решении используются три переменные: `low`, `mid` и `high`. Пока `low` меньше или равно `high`, выполняются следующие шаги.

1. Вычислить `mid` как результат целочисленного деления `high + low` на 2.
2. Если искомое значение больше элемента входного списка в позиции `mid`, то игнорировать левую половину списка.
3. Если искомое значение меньше элемента входного списка в позиции `mid`, то игнорировать правую половину списка.
4. Если искомое значение не больше и не меньше элемента входного списка в позиции `mid`, значит, мы нашли искомое значение, поэтому возвращаем индекс `mid`.

Если цикл завершился без возврата значения, значит, в списке нет искомого значения, и мы возвращаем –1.

Задача 44

```
def quicksort(input_list: list[int], low: int, high: int)
    -> list[int]:

    if low < high:
        pivot_idx = partition(input_list, low, high)
        quicksort(input_list, low, pivot_idx - 1)
        quicksort(input_list, pivot_idx + 1, high)

    return input_list

def partition(input_list: list[int], low: int, high: int)
    -> int:

    pivot = input_list[high]
    pivot_idx = low - 1

    for current_idx in range(low, high):

        if input_list[current_idx] <= pivot:
            pivot_idx = pivot_idx + 1
            input_list[pivot_idx], input_list[current_idx] = (
                input_list[current_idx],
                input_list[pivot_idx],
            )

    input_list[pivot_idx + 1], input_list[high] = (
        input_list[high],
        input_list[pivot_idx + 1],
    )

    return pivot_idx + 1
```

Здесь `quicksort` – главная функция, выполняющая сортировку. Она принимает входной список и два индекса, `low` и `high`, определяющих, какую часть списка сортировать, и пользуется функцией `partition`, чтобы разделить список на две части: слева и справа от опорного элемента `pivot`, а затем рекурсивно вызывает себя, чтобы отсортировать оба подсписка.

Функция `partition` реализует следующую схему разбиения Ломуто:

1. Выбрать последний элемент списка в качестве опорного.
2. Инициализировать переменную `pivot_idx` значением `low - 1`.
3. Обойти элементы в диапазоне от `low` до `high - 1` (текущий элемент обозначается `current_idx`).

4. Если элемент `input_list[current_idx]` меньше или равен `pivot`, обменять опорный элемент с `input_list[pivot_idx + 1]` и увеличить `pivot_idx` на 1.
5. По завершении цикла обменять опорный элемент с `input_list[pivot_idx + 1]`.
6. Вернуть `pivot_idx + 1` в качестве индекса опорного элемента.

Функция `partition` возвращает индекс опорного элемента после разбиения `input_list` на две части. Функция `quicksort` использует этот индекс, чтобы определить, какие два подсписка должны быть отсортированы на следующей итерации. Алгоритм завершается, когда `low >= high`, что означает, что в подсписке больше нет элементов и он уже отсортирован.

Средняя временная сложность этой реализации составляет $O(n \log n)$, что делает ее эффективным алгоритмом сортировки для больших списков. Пространственная сложность реализации составляет $O(\log n)$, потому что используется подход «разделяй и властвуй», который требует логарифмического объема дополнительной памяти в стеке вызовов.

Лучший способ разобраться в этом решении – пройти его под отладчиком в пошаговом режиме. Поставьте точку останова на первой строчке функции `quicksort` и исполняйте код по шагам, чтобы увидеть, как он работает. Не делайте входные данные слишком сложными, чтобы не запутаться.

Задача 45

```
def solve_knapsack_problem(
    items: list[tuple[int, int]], knapsack_capacity: int
) -> int:
    # Начать рекурсивный вызов с последнего предмета
    return knapsack_recursive(items, len(items), knapsack_capacity)

def knapsack_recursive(
    items: list[tuple[int, int]],
    index: int,
    remaining_capacity: int
) -> int:
        # Базовый случай: если мы рассмотрели все предметы или вместимость
        # равна нулю
        if index == 0 or remaining_capacity == 0:
            return 0

        # Получить вес и ценность текущего предмета
        weight, value = items[index - 1]

        # Если вес текущего предмета больше оставшейся вместимости,
        # то положить его в рюкзак нельзя
        if weight > remaining_capacity:
            return knapsack_recursive(
                items, index - 1, remaining_capacity
        )

        # В противном случае у нас есть выбор:
        # 1. Положить текущий предмет и рекурсивно проверить оставшуюся
        # вместимость
        include_current = value + knapsack_recursive(
            items, index - 1, remaining_capacity - weight
        )

        # 2. Не класть текущий предмет и рекурсивно проверить следующий
        # предмет
        exclude_current = knapsack_recursive(items, index - 1,
                                             remaining_capacity)

        # Вернуть максимум из двух вариантов
        return max(include_current, exclude_current)
```

Здесь для решения задачи об укладке рюкзака используется рекурсия, на каждом шаге которой текущий предмет укладывается или не укладывается в рюкзак. По завершении рекурсии возвращается максимальное значение.

Примечание: временная сложность этого решения составляет $O(2^n)$, поэтому оно не удовлетворяет требованиям дополнительной задачи 45.1.

Задача 45.1 (дополнительная)

```
def solve_knapsack_problem_bonus(
    items: list[tuple[int, int]], knapsack_capacity: int
) -> int:
    # Создать матрицу для хранения максимального значения при каждом весе
    max_value_matrix = [
        [0 for _ in range(knapsack_capacity + 1)]
            for _ in range(len(items) + 1)
    ]

    for item_idx in range(1, len(items) + 1):
        for weight in range(1, knapsack_capacity + 1):

            if items[item_idx - 1][0] > weight:
                max_value_matrix[item_idx][weight]
                    = max_value_matrix[item_idx - 1][weight]
            else:
                max_value_matrix[item_idx][weight] = max(
                    max_value_matrix[item_idx - 1][weight],
                    max_value_matrix[item_idx - 1]
                        [weight - items[item_idx - 1][0]]
                        + items[item_idx - 1][1],
                )

    return max_value_matrix[len(items)][knapsack_capacity]
```

Самый эффективный по времени алгоритм решения задачи об укладке рюкзака основан на динамическом программировании, а точнее на подходе «снизу вверх». Посмотрим, как он работает.

1. Создается двумерная матрица `max_value_matrix` размерности `(len(items) + 1) x (knapsack_capacity + 1)`. В ней будут храниться максимальные ценности, достижимые при каждом весе. Матрица инициализируется нулями посредством спискового включения. Строки представляют предметы, а столбцы – веса.
2. В двух вложенных циклах обходятся все комбинации предметов и весов, начиная с первого предмета и веса 1.
3. Во внутреннем цикле функция сравнивает вес текущего предмета (`items[item_idx - 1][0]`) с текущим рассматриваемым весом. Если текущий предмет весит больше, значит, его нельзя положить в рюкзак с такой вместимостью. Следовательно, максимальная ценность остается такой же, как для предыдущего предмета, поэтому она копируется из преды-

дущей строки матрицы: `max_value_matrix[item_idx][weight] = max_value_matrix[item_idx - 1][weight]`.

4. Если вес текущего предмета меньше или равен текущему рассматриваемому весу, значит, его можно положить в рюкзак с такой вместимостью. В этом случае функция вычисляет два значения:
   a) максимальную ценность, получаемую, если не класть текущий предмет. Она берется из предыдущей строки матрицы: `max_value_matrix[item_idx - 1][weight]`;
   b) максимальную ценность, получаемую, если положить текущий предмет. Она вычисляется путем прибавления ценности текущего предмета (`items[item_idx - 1][1]`) к максимальной ценности, получаемой для разности весов (`weight - items[item_idx - 1][0]`). Эта разность представляет вместимость, которая остается после укладывания текущего предмета в рюкзак;
   c) максимум из двух вычисленных значений записывается в элемент `max_value_matrix[item_idx][weight]`.
5. После перебора всех предметов и весов функция возвращает значение, хранящееся в элементе `max_value_matrix[len(items)][knapsack_capacity]` и равное максимальной ценности, достижимой при заданном ограничении на вес.

Временная сложность этого алгоритма равна $O(nW)$, где n – число предметов, а W – максимальный вес.

Чтобы было понятнее, рассмотрим, как выглядит конечная матрица для следующих входных данных:

```
items = [(5, 2), (1, 1000), (100, 1), (25, 25), (2, 1000)]
knapsack_capacity = 5

max_value_matrix = [
    [0,    0,    0,    0,    0,    0]
    # Для weight = 5 мы можем включить (5, 2)
    [0,    0,    0,    0,    0,    2]
    # Для weight >= 1 мы можем включить (1, 1000)
    [0, 1000, 1000, 1000, 1000, 1000]
    # Не можем включить (100, 1), потому что weight > knapsack_capacity
    [0, 1000, 1000, 1000, 1000, 1000]
    # Не можем включить (25, 25), потому что weight > knapsack_capacity
    [0, 1000, 1000, 1000, 1000, 1000]
    # Для weight >= 2 мы можем включить (2, 1000)
    [0, 1000, 1000, 2000, 2000, 2000]
```

- Каждый столбец матрицы представляет вес: первый столбец соответствует весу 0, второй – весу 1 и т. д.
- Каждая строка матрицы представляет предмет: в первой строке все элементы равны 0, во второй все они равны `items[0]` и т. д.
- Чтобы получить максимально возможную ценность, мы возвращаем значение в правом нижнем углу матрицы.

Задача 46

```
import socket
import struct

def ip_range_to_list(input_ip_range: str) -> list[str]:
    start, end = input_ip_range.split("-")
    start_ip = struct.unpack("!L", socket.inet_aton(start))[0]
    end_ip = struct.unpack("!L", socket.inet_aton(end))[0]
    return [
        socket.inet_ntoa(struct.pack("!L", ip_address))
        for ip_address in range(start_ip, end_ip + 1)
    ]
```

Здесь используется метод `split`, чтобы выделить из входной строки начальный и конечный IP-адреса. Затем с помощью вызова `struct.unpack("!L", socket.inet_aton(ip))` IP-адреса преобразуются в 32-разрядные целые числа.

После этого списковое включение и вызов `range(start_ip, end_ip + 1)` используются, чтобы сгенерировать диапазон целых чисел, соответствующий IP-адресам в этом диапазоне. И наконец, вызов `socket.inet_ntoa(struct.pack("!L", ip_address))` преобразует целые числа обратно в IP-адреса в формате `"x.x.x.x"`.

Это решение эффективнее по времени, потому что поразрядные операции быстрее операций над строками.

Отметим, что в решении поддерживаются только IPv4-адреса.

Рассмотрим пример для `input_ip_range = "192.168.1.1-192.168.1.5"`.

```
start="192.168.1.1"
end="192.168.1.5"
start_ip = struct.unpack("!L", socket.inet_aton(start))[0]
 = struct.unpack("!L", b"\xc0\xa8\x01\x01")[0]
 = (3232235777,)[0]
 = 3232235777

end_ip = struct.unpack("!L", socket.inet_aton(end))[0]
 = struct.unpack("!L", b"\xc0\xa8\x01\x05")[0]
 = (3232235781,)[0]
 = 3232235781

[ socket.inet_ntoa(struct.pack("!L", ip_address))
    for ip_address in range(start_ip, end_ip + 1) ]

 = [socket.inet_ntoa(struct.pack("!L", ip_address))
    for ip_address in [3232235777, 3232235778, 3232235779
                       3232235780, 3232235781]]

 = ["192.168.1.1", "192.168.1.2", "192.168.1.3", "192.168.1.4", "192.168.1.5"]
```

Задача 47

```
def solve_maze(
    maze: list[list[int]], start_pos: tuple[int, int], end_pos:
tuple[int, int]
) -> bool:

    num_rows, num_cols = len(maze), len(maze[0])
    queue = []
    queue.append(start_pos)
    maze[start_pos[1]][start_pos[0]] = 2

    while queue:
        curr_x, curr_y = queue.pop(0)

        if curr_x == end_pos[0] and curr_y == end_pos[1]:
            return True

        for exploring_x, exploring_y in [
            (curr_x + 1, curr_y),
            (curr_x - 1, curr_y),
            (curr_x, curr_y + 1),
            (curr_x, curr_y - 1),
        ]:
            if (
                0 <= exploring_x < num_cols
                and 0 <= exploring_y < num_rows
                and maze[exploring_y][exploring_x] == 0
            ):
                queue.append((exploring_x, exploring_y))
                maze[exploring_y][exploring_x] = 2

    return False
```

Здесь для нахождения пути из начальной позиции в конечную используется алгоритм поиска в ширину. Мы начинаем с начальной позиции и исследуем все возможные пути в конечную позицию. Посещенные позиции запоминаются, а значение в них изменяется с 0 на 2, чтобы избежать посещения одной и той же позиции. Если конечная позиция достигнута, функция возвращает `True`, иначе `False`.

Разберем решение по шагам.

1. Переменные `num_rows` и `num_cols` инициализируются числом строк и столбцов лабиринта соответственно.

2. Создается список `queue`, используемый как очередь позиций, подлежащих исследованию. Первоначально в очередь помещается позиция `start_pos`.
3. Начальная позиция помечается как посещенная: в `maze[start_pos[1]][start_pos[0]]` записывается 2.
4. Начинается главный цикл, который продолжается, пока очередь не опустеет.
   a. Получаем текущую позицию, извлекая ее из начала очереди с помощью вызова `queue.pop(0)`. В переменные `curr_x` и `curr_y` записываются координаты текущей позиции.
   b. Если текущая позиция совпадает с `end_pos`, значит, цель достигнута, и функция возвращает `True`.
   c. В противном случае исследуются позиции, соседние с текущей, в порядке справа, слева, снизу и сверху. Координаты этих позиций сохраняются в переменных `exploring_x` и `exploring_y`.
   d. Для каждой соседней позиции проверяется, находится ли она внутри лабиринта (`0 <= exploring_x < num_rows` и `0 <= exploring_y < num_cols`) и разрешен ли переход в нее (`maze[exploring_y][exploring_x] == 0`).
      i. Если соседняя позиция допустима, то она помещается в очередь для исследования в будущем и помечается как посещенная путем записи 2 в `maze[exploring_y][exploring_x]`.
5. Если цикл завершился, а цель так и не достигнута, значит, пути из начальной позиции в конечную не существует, и функция возвращает `False`.

Задача 48

```
from __future__ import annotations
from typing import Iterator

class TreeNode:
    def __init__(
        self,
        val: int = 0,
        left: TreeNode | None = None,
        right: TreeNode | None = None,
    ) -> None:
        self.val = val
        self.left = left
        self.right = right

def traverse_inorder(root_node: TreeNode | None)
    -> Iterator[int]:
    if root_node is None:
        return []
    yield from traverse_inorder(root_node.left)
    yield root_node.val
    yield from traverse_inorder(root_node.right)
```

Здесь рекурсия используется для обхода двоичного дерева в симметричном порядке.

1. Сначала определим базовый случай: `if root_node is None`. Это означает, что дерево пусто, поэтому возвращается пустой список.
2. Если `root_node` не равен `None`, то рекурсивный обход продолжается.
   a. Сначала функция рекурсивно обходит левое поддерево, вызывая `traverse_inorder(root_node.left)`. Ключевое слово `yield from` отдает значения, возвращенные в результате рекурсивного обращения к `traverse_inorder(root_node.left)`.
   b. Отдав все значения из левого поддерева, функция отдает значение текущего узла (`yield root_node.val`).
   c. И наконец, функция рекурсивно обходит правое поддерево, вызывая `traverse_inorder(root_node.right)`. Снова ключевое слово `yield from` отдает значения, возвращенные в результате рекурсивного обращения к `traverse_inorder(root_node.right)`.

Рекурсия продолжается, пока не достигнет листьев дерева (в этот момент `root_node` становится равен `None`), а затем возвращается по своим следам, отдавая значения в порядке, определяемом симметричным обходом.

Задача 48.1 (дополнительная)

```
from __future__ import annotations
from typing import Iterator

def morris_traverse_inorder(root_node: TreeNode | None)
    -> Iterator[int]:

    current_node = root_node

    while current_node is not None:

        if current_node.left is None:
            yield current_node.val
            current_node = current_node.right

        else:
            predecessor_node = current_node.left

            while predecessor_node.right is not None
                and predecessor_node.right is not current_node:
                predecessor_node = predecessor_node.right

            if predecessor_node.right is None:
                predecessor_node.right = current_node
                current_node = current_node.left
            else:
                predecessor_node.right = None
                yield current_node.val
                current_node = current_node.right
```

Здесь функция принимает аргумент `root_node`, представляющий корень двоичного дерева, и возвращает итератор, который отдает значения узлов в симметричном порядке обхода.

1. В самом начале переменная `current_node` инициализируется значением `root_node`.
2. Функция входит в цикл, который продолжается, пока `current_node` не станет равным `None`.
3. В цикле проверяется, есть ли у `current_node` левый потомок.
   a. Если нет, значит, это самый левый узел в текущем поддереве, поэтому функция отдает его значение и переходит к правому потомку текущего узла (`current_node = current_node.right`).
   b. Иначе предшествующий узел (`predecessor_node`) делается равным левому потомку текущего узла. Затем проверяется, есть ли у предшествующего узла правый потомок и не совпадает ли он с `current_node`. Смысл этого условия в том,

чтобы проверить, не посещался ли предшествующий узел ранее. Цикл продолжается, пока не найдет подходящего предшественника или не дойдет до конца правого поддерева предшественника.

   i. Если у предшественника нет правого потомка, значит, он еще не посещался. Функция делает правым потомком предшественника `current_node`, чтобы установить между ними связь (`predecessor_node.right = current_node`). Эта связь позволит коду вернуться в текущий узел после завершения симметричного обхода поддерева предшественника. Затем функция переходит к левому потомку `current_node` (`current_node = current_node.left`), чтобы продолжить обход левого поддерева.
   ii. Если у предшественника уже имеется правый потомок (он посещался ранее), то функция удаляет связь, делая указатель на правого потомка предшественника равным `None` (`predecessor_node.right = None`). Функция отдает значение `current_node`, потому что завершила симметричный обход его левого поддерева. Наконец, функция переходит к правому потомку `current_node` (`current_node = current_node.right`), чтобы обойти правое поддерево.

4. Цикл продолжается, пока все узлы двоичного дерева не будут посещены и отданы.

Задача 49

```
def solve_climbing_stairs_problem(total_stairs: int) -> int:
    if total_stairs == 0:
        return 0
    if total_stairs == 1:
        return 1
    if total_stairs == 2:
        return 2

    num_combinations = [0] * (total_stairs + 1)
    num_combinations[0] = 0
    num_combinations[1] = 1
    num_combinations[2] = 2

    for num_stairs in range(3, total_stairs + 1):
        num_combinations[num_stairs] = (
            num_combinations[num_stairs - 1]
                + num_combinations[num_stairs - 2]
        )
    return num_combinations[-1]
```

Сначала проверяются базовые случаи, когда `total_stairs` равно 0, 1 или 2. Если `total_stairs` равно 0, значит, ступенек нет, поэтому функция возвращает 0. Если `total_stairs` равно 1, то имеется только один способ подняться по лестнице (на одну ступеньку), поэтому функция возвращает 1. Если `total_stairs` равно 2, то имеется два способа подняться по лестнице (либо на две ступеньки сразу, либо два раза по одной ступеньке), поэтому функция возвращает 2.

Если `total_stairs` больше 2, то функция создает список `num_combinations` с `total_stairs + 1` элементами, инициализируя их все нулями. В этом списке для каждого количества ступеней будет храниться число способов подняться по лестнице.

- Элементам `num_combinations[0]`, `[1]` и `[2]` присваиваются значения 0, 1 и 2 соответственно. Эти значения соответствуют базовым случаям, когда `total_stairs` равно 0, 1, 2.

Затем следует цикл вычисления количества способов подъема для каждого числа ступенек от 3 до `total_stairs + 1`. Функция обходит этот диапазон и вычисляет число способов для текущего количества ступенек, складывая два предыдущих значения из списка `num_combinations`.

В самом конце функция возвращает последний элемент списка `num_combinations`, в котором находится число способов подняться на лестницу с `total_stairs` ступенями.

В коде используется динамическое программирование: чтобы избежать избыточных вычислений, ранее вычисленные значения сохраняются и используются повторно. Такой подход повышает эффективность решения.

Задача 49.1 (дополнительная)

```
def solve_climbing_stairs_problem_with_output(
    total_stairs: int)
-> list[list[int]]:

    if total_stairs == 0:
        return []
    if total_stairs == 1:
        return [[1]]
    if total_stairs == 2:
        return [[1, 1], [2]]

    num_combinations = [[] for _ in range(total_stairs + 1)]
    num_combinations[0] = []
    num_combinations[1] = [[1]]
    num_combinations[2] = [[1, 1], [2]]

    for num_stairs in range(3, total_stairs + 1):
        for seq in num_combinations[num_stairs - 1]:
            num_combinations[num_stairs].append(seq + [1])
        for seq in num_combinations[num_stairs - 2]:
            num_combinations[num_stairs].append(seq + [2])

    return num_combinations[-1]
```

Здесь используется такой же подход, как и раньше, но вместо одного списка для хранения числа способов подняться на лестницу создается список списков, в котором каждый подсписок представляет один способ подъема. Функция инициализирует первые три элемента списками `[]`, `[[1]]` и `[[1,1], [2]]` – это единственные способы подняться на лестницу с 0, 1 и 2 ступенями.

Далее используются вложенные циклы для заполнения остальных элементов списка списков. Внешний цикл организован по количеству ступеней, а во внутренних функция производит обход последовательностей, созданных на предыдущем шаге, и в каждую последовательность добавляет 1 или 2, создавая тем самым новые последовательности способов подняться по лестнице. В конце функция возвращает последний элемент списка списков, т. е. список всех возможных способов подняться по лестнице.

Примечание: временная сложность этого модифицированного решения составляет $O(2^n)$, а пространственная – $O(n*2^n)$, потому что требуется хранить все возможные способы подняться по лестнице, число которых растет экспоненциально.

Задача 49.2 (дополнительная)

```
def solve_climbing_stairs_problem_with_three_steps_allowed(
    total_stairs: int
) -> int:

    if total_stairs == 0:
        return 0
    if total_stairs == 1:
        return 1
    if total_stairs == 2:
        return 2
    if total_stairs == 3:
        return 4

    num_combinations = [0] * (total_stairs + 1)
    num_combinations[0] = 0
    num_combinations[1] = 1
    num_combinations[2] = 2
    num_combinations[3] = 4

    for num_stairs in range(4, total_stairs + 1):
        num_combinations[num_stairs] = (
            num_combinations[num_stairs - 1]
            + num_combinations[num_stairs - 2]
            + num_combinations[num_stairs - 3]
        )

    return num_combinations[total_stairs]
```

Видно, что это решение очень похоже на предыдущее; нужно лишь внести небольшие изменения, учитывающие возможность подниматься на одну, две или три ступеньки за один шаг.

1. Новый базовый случай, когда `total_stairs` равно 3. В этом случае мы возвращаем 4, потому что есть четыре способа подняться по лестнице: (`[1,1,1]`, `[1,2]`, `[2,1]`, `[3]`).
2. Новый элемент в списке: `num_combinations[3] = 4`.
3. Новое слагаемое при вычислении элемента `num_combinations[num_stairs]` (`num_combinations[num_stairs - 3]`).

Задача 50

```
def solve_coin_change_problem(
    coin_values: list[int],
    target_amount: int
) -> int:

    if target_amount == 0:
        return 0

    if coin_values == []:
        return -1

    min_num_coins = [float("inf")] * (target_amount + 1)
    min_num_coins[0] = 0

    for current_target_amount in range(1, target_amount + 1):
        for current_coin_value in coin_values:

            if current_target_amount - current_coin_value >= 0:
                min_num_coins[current_target_amount] = min(
                    min_num_coins[current_target_amount],
                    min_num_coins[current_target_amount
                        - current_coin_value] + 1,
                )

    return int(min_num_coins[target_amount])
```

Сначала проверяются два базовых случая.

1. Если `target_amount` равно 0, то для получения целевой суммы нам вообще не нужны монеты, поэтому функция возвращает 0.
2. Если список `coin_values` пуст, значит, монет не осталось, поэтому разменять целевую сумму невозможно. В таком случае функция возвращает –1, показывая, что размен невозможен.

Функция создает список `min_num_coins` с `target_amount + 1` элементами, инициализируя их значением бесконечность (`float("inf")`). В этом списке для каждой целевой суммы от 0 до `target_amount` будет храниться минимальное количество монет.

В элемент `min_num_coins[0]` записывается 0, потому что для размена суммы 0 вообще не нужны монеты.

Для вычисления минимального числа монет, необходимого для размена каждой целевой суммы от 1 до `target_amount`, используются вложенные циклы. Во внешнем цикле обходится диапазон от 1

до `target_amount + 1`, представляющий текущую целевую сумму. Во внутреннем цикле обходится список `coin_values` и рассматривается потенциальное участие монет каждого достоинства в размене.

1. Во внутреннем цикле проверяется, приведет ли вычитание текущего достоинства монеты из текущей целевой суммы (`current_target_amount - current_coin_value`) к неотрицательной величине. Если да, значит, монеты текущего достоинства могут участвовать в размене.
2. Функция изменяет элемент `min_num_coins[current_target_amount]`, записывая в него минимум из его текущего значения и суммы `min_num_coins[current_target_amount - current_coin_value] + 1`. Это минимальное число монет, необходимых для размена текущей целевой суммы с учетом текущей монеты.

По завершении внешнего цикла функция возвращает целое число, хранящееся в элементе `min_num_coins[target_amount]`, поскольку оно равно минимальному числу монет, необходимому для размена целевой суммы.

В коде применяется динамическое программирование: ранее вычисленные значения сохраняются в списке `min_num_coins` и используются повторно. Это позволяет избежать избыточных вычислений и эффективно найти оптимальное решение.

Шуточные задачи: решения

Обратный поиск в DNS

```
import socket

def reverse_dns_lookup(ip_address: str) -> str:
    try:
        domain_name = socket.gethostbyaddr(ip_address)[0]
        return domain_name
    except socket.herror:
        return None
```

Здесь обратный поиск в DNS производится с помощью функции `socket.gethostbyaddr()`, которая принимает IP-адрес и возвращает кортеж, содержащий доменное имя и дополнительную информацию.

Если обратный поиск в DNS завершился успешно, то функция извлекает из кортежа доменное имя и возвращает его в виде строки.

Если же во время поиска возникло исключение типа `socket.herror`, то функция обрабатывает его и возвращает `None`, сообщая тем самым, что для заданного IP-адреса не удалось найти доменное имя.

Для использования этой функции необходимо импортировать модуль `socket`, который предоставляет функциональность для работы с сетью, в т. ч. поиска в DNS.

Матрешки

```
from __future__ import annotations

class RussianDoll:
    def __init__(
            self,
            size: int,
            colour: str,
            child_doll: RussianDoll | None = None
    ) -> None:

        self.size = size
        self.colour = colour
        self.child_doll = child_doll

    def get_number_of_children(self) -> int:
        if self.child_doll is None:
            return 0
        return 1 + self.child_doll.get_number_of_children()

    def print_unpack(self) -> None:
        print(
            f"Unpacking a {self.colour} doll of size:
              {self.size} with {self.get_number_of_children()}
               nested dolls inside."
        )

def unpack_dolls(doll: RussianDoll) -> int:
    print(f"Total number of dolls unpacked:
            {inner_unpack_dolls(doll)}")

def inner_unpack_dolls(doll: RussianDoll) -> int:

    doll.print_unpack()

    if doll.child_doll is None:
        return 1

    return 1 + inner_unpack_dolls(doll.child_doll)
```

В этом решении используются три вспомогательные функции в дополнение к приведенной в заготовке кода.

Функция `inner_unpack_dolls` делает несколько вещей.

1. Открывает текущую куклу, вызывая метод `print_unpack`, который печатает на консоли строку «Unpacking a {self.colour} doll of size: {self.size} with {self.get_number_of_children()} nested dolls inside».

   a. Здесь вызывается метод `get_number_of_children`, который рекурсивно определяет, сколько кукол находится внутри данной.
2. Проверяет, есть ли кукла внутри данной, и если нет, то возвращает 1 (это базовый случай рекурсии).
3. Рекурсивно вызывает себя, что открыть куклы, спрятанные внутри текущей. При этом не только открываются куклы, но и подсчитывается, сколько всего кукол было открыто. Затем это значение возвращается функции `unpack_dolls`.

Фрактальное дерево

```
import turtle

def draw_tree(depth: int) -> None:
    if depth == 0:
        return
    turtle.forward(20*depth)
    turtle.left(30)
    draw_tree(depth-1)
    turtle.right(60)
    draw_tree(depth-1)
    turtle.left(30)
    turtle.backward(20*depth)

draw_tree(5)
turtle.exitonclick()
```

В этом коде используется модуль `turtle` для рисования древовидного узора в графическом окне. Далее следует пошаговое объяснение работы функции.

1. Проверяется базовый случай – `depth == 0`.
   a. Если это так, значит, рекурсия достигла желаемой глубины, и функция просто возвращает управление, не совершая дальнейших рекурсивных вызовов.
2. Если базовый случай еще не наступил, то функция продолжает работу. Черепашка продвигается вперед на расстояние `20 * depth` единиц. Параметр `depth` управляет длиной каждой ветви в зависимости от глубины.
3. Затем черепашка поворачивает на 30 градусов влево, меняя направление движения.
4. Далее рекурсивно вызывается функция `draw_tree` с уменьшенным значением глубины (`depth-1`). Это создает эффект ветвления по мере расширения дерева. Рекурсивный вызов отвечает за рисование поддеревьев слева и справа от текущей ветви.
5. После рекурсивного вызова черепашка поворачивает на 60 градусов вправо, готовясь к рисованию следующей ветви.
6. Снова рекурсивно вызывается `draw_tree` с уменьшенной глубиной, чтобы нарисовать следующую ветвь в противоположном направлении.

7. Черепашка поворачивает на 30 градусов влево, чтобы восстановить исходную ориентацию.
8. Наконец, черепашка движется назад на то же расстояние, на какое раньше продвинулась вперед (`20 * depth` единиц), возвращаясь в начало текущей ветви.

Функция `turtle.exitonclick()` вызывается, для того чтобы графическое окно оставалось открытым, пока пользователь не щелкнет мышью.

Пинг-понг

Решение слишком длинное, чтобы приводить его на страницах книги, но его можно найти в репозитории на github по короткой ссылке *https://rb.gy/6n2rq*.

В предположении, что вы открыли файл с решением, ниже приводится его описание.

1. В самом начале импортируются необходимые модули: `pygame` для разработки игры и `sys` для получения доступа к системно-зависимым параметрам и функциям. Библиотека pygame инициализируется методом `pygame.init()`.
2. Определяется несколько констант, в т. ч. ширина и высота игрового окна, число кадров в секунду, размеры и скорость перемещения ракеток, а также различные пороговые значения и размеры, относящиеся к мячу и подсчету очков.
3. Заводятся переменные для хранения очков и начальных положений ракеток и мяча.
4. Создается игровое окно методом `pygame.display.set_mode()`, которому передаются ширина и высота. Задается текст заголовка окна методом `pygame.display.set_caption()`.
5. Создаются обе ракетки (`left_paddle` и `right_paddle`) в виде объектов `pygame.Rect`. Эти прямоугольники представляют положения и размеры ракеток.
6. Начинается главный цикл игры, который продолжается, пока игрок не захочет выйти. В этом цикле производятся следующие действия:
   a) методом `pygame.event.get()` обрабатываются события. Если возникло событие `pygame.QUIT` (например, пользователь закрыл окно игры), то игра завершается с помощью вызовов `pygame.quit()` и `sys.exit()`;
   b) проверяется, изменил ли пользователь положение ракеток. Для этого вызывается метод `pygame.key.get_pressed()`, который проверяет нажатие определенных клавиш (например, W, S, UP, DOWN). В зависимости от нажатой клавиши положение ракетки изменяется с учетом заданных ограничений на верхнюю и нижнюю границы игрового поля;
   c) обновляется положение мяча путем прибавления вектора `ball_speed` к текущему положению;

d) проверяется, столкнулся ли мяч с ракеткой. Для этого функция проверяет, что координата *x* мяча находится на одной линии с ракеткой, а координата *y* мяча принадлежит диапазону вертикальных координат ракетки. Если имеет место столкновение и мяч двигался в сторону ракетки, то скорость мяча в направлении *x* меняется на противоположную;
e) проверяется, коснулся ли мяч верхней или нижней границы экрана, и если да, то скорость мяча в направлении *y* меняется на противоположную, чтобы вернуть мяч в игру;
f) координата *x* мяча сравнивается с порогом начисления очков. Если мяч ушел за левый порог, то очко присуждается правому игроку, и мяч возвращается в исходное положение. Если мяч ушел за правый порог, то очко присуждается левому игроку, и мяч возвращается в исходное положение;
g) игровой экран закрашивается черным цветом с помощью метода `screen.fill()`. Затем на экране рисуются ракетки и мяч с помощью `pygame.draw.rect()` и `pygame.draw.circle()` соответственно;
h) текущий счет формируется в виде текста указанным шрифтом заданного размера с помощью вызова `FONT.render()`. Затем сформированный текст отображается на экране методом `screen.blit()`;
i) игровой экран обновляется методом `pygame.display.update()`, а частота кадров определяется методом `clock.tick(FPS)`, который отвечает за смену кадров указанное число раз в секунду. Цикл продолжается, и на каждой итерации состояние игры обновляется.

Пинг-понг (дополнение)

Решение слишком длинное, чтобы приводить его на страницах книги, но его можно найти в репозитории на GitHub по короткой ссылке *https://rb.gy/p951y*.

В предположении, что вы открыли файл с решением, ниже приводится его описание.

1. В самом начале импортируются необходимые модули: `pygame`, `sys`, `math` и `random`. Библиотека pygame инициализируется методом `pygame.init()`.
2. Определяются два класса: `StaticColour` и `ChangingColour`. Они представляют различные типы цветов, допустимых в игре. В `StaticColour` хранится фиксированное значение цвета, а `ChangingColour` может пробегать список цветов по кругу. В классе `ChangingColour` используется функция `pygame.time.set_timer()`, которая раз в 500 мс генерирует пользовательское событие (`pygame.USEREVENT`), используемое, чтобы периодически изменять цвет.
3. Определяется класс `ColourPicker`, упрощающий выбор цвета в игре. Он включает предопределенные цвета и позволяет игроку выбирать цвета различных игровых элементов: фона, ракеток и мяча. Метод `select_colour()` – главная точка входа для выбора цвета. Он открывает окно, в котором игрок может выбрать один из доступных цветов, щелкнув по цветному квадратику. Метод возвращает выбранный цвет.
4. Класс `Ball` представляет мяч. У него имеются атрибуты для положения, скорости, размера и цвета. Класс предоставляет методы для перемещения мяча, возврата мяча в исходное положение и изменения направления его движения. Метод `get_colour()` возвращает текущий цвет мяча.
5. Класс `Paddle` представляет ракетку. У него имеются атрибуты для положения, скорости, размеров и цвета. Класс предоставляет методы для перемещения ракетки вверх и вниз, получения ее цвета и преобразования ракетки в объект `pygame.Rect`.
6. Класс `PingPong` инкапсулирует логику игры и служит главной точкой входа. В нем определены различные константы для размера экрана, свойств мяча и ракетки, а также настроек игры.

7. В конструкторе класса `PingPong` методом `pygame.display.set_mode()` создается игровое окно и задается его заголовок. Игроку предлагается выбрать цвета фона, ракеток и мяча с помощью класса `ColourPicker`.
8. Метод `create_balls()` инициализирует мячи для игры. Создается несколько экземпляров класса `Ball` со случайными начальными скоростями и положениями.
9. Метод `play()` реализует главный цикл игры. Он непрерывно обрабатывает события, обновляет состояние игры и отрисовывает экран.
   a. Метод `handle_event()` обрабатывает различные события, например выход из игры, нажатия кнопок мыши для выбора цвета и пользовательское событие `pygame.USEREVENT` для периодического изменения цвета. Если игровые объекты имеют цвета типа `ChangingColour`, то он обновляет их.
   b. Метод `handle_paddle_movement()` обрабатывает перемещение ракетки в зависимости от действий пользователя – нажатия той или иной клавиши.
   c. Метод `ball_is_hit_by_paddle()` проверяет, столкнулся ли мяч с ракеткой. Для этого он анализирует положения и размеры ракеток и мяча.
   d. Метод `ball_touched_top_or_bottom_of_screen()` проверяет, коснулся ли мяч верхней или нижней границы экрана, и если да, меняет направление движения по вертикали на противоположное.
   e. Методы `left_scoring_threshold_hit()` и `right_scoring_threshold_hit()` проверяют, пересек ли мяч пороговую линию начисления очков слева или справа от экрана, и если да, начисляет очко игроку на противоположной стороне.
   f. Метод `draw()` отвечает за рисование игровых объектов на экране. Он закрашивает экран цветом фона, рисует ракетки и мячи и отображает текущий цвет.

Наконец, создается экземпляр класса `PingPong()` и вызывается его метод `play()`, чтобы начать игру.

Средство рисования

```
import pygame
import sys

pygame.init()

class DrawingTool:

    SCREEN_WIDTH, SCREEN_HEIGHT = 800, 600
    PAINT_SIZE = 10

    def __init__(self):

        # Создать окно
        self.screen = pygame.display.set_mode(
            (DrawingTool.SCREEN_WIDTH,
             DrawingTool.SCREEN_HEIGHT),
        )
        pygame.display.set_caption("Drawing Tool")

        self.is_drawing = False

    def start(self):
        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                if event.type == pygame.MOUSEBUTTONDOWN:
                    self.is_drawing = True
                if event.type == pygame.MOUSEBUTTONUP:
                    self.is_drawing = False

            if self.is_drawing:
                self.draw(pygame.mouse.get_pos())

            pygame.display.update()

    def draw(self, pos):
        pygame.draw.circle(
            self.screen,
            (255, 255, 255),
            pos,
            DrawingTool.PAINT_SIZE)

if __name__ == "__main__":
    DrawingTool().start()
```

Разберем это решение по шагам, чтобы понять, как оно работает.

1. Вначале импортируются необходимые модули: `pygame` для разработки игры и `sys` для получения доступа к системно-зависимым параметрам и функциям. Библиотека pygame инициализируется методом `pygame.init()`.
2. Определяется класс `DrawingTool`, инкапсулирующий функциональность средства рисования. В этом классе определено несколько констант:
   a) `SCREEN_WIDTH` и `SCREEN_HEIGHT` представляют размеры окна рисования;
   b) `PAINT_SIZE` представляет размер кисти для рисования.
3. Метод `__init__` класса `DrawingTool` инициализирует окно рисования и задает заголовок окна. Он также инициализирует переменную `is_drawing` значением `False`, которое сообщает, что в настоящий момент пользователь не рисует.
4. Метод `start` содержит главный цикл программы. Он работает, пока программа не завершится. В нем обрабатываются события, инициированные пользователем (например, щелчки мышью и закрытие окна).
   a. Если пользователь нажимает кнопку закрытия окна, то вызывается метод `pygame.quit()`, завершающий игру, и метод `sys.exit()`, который осуществляет выход из программы.
   b. Когда пользователь нажимает кнопку мыши, переменной `self.is_drawing` присваивается значение `True`, показывающее, что пользователь начал рисовать. Когда пользователь отпускает кнопку мыши, переменная `self.is_drawing` сбрасывается в `False`, показывающее, что пользователь прекратил рисовать. Кроме того, если флаг `is_drawing` равен `True`, то вызывается метод `draw`, которому передается положение мыши, полученное от метода `pygame.mouse.get_pos()`.
5. Метод `draw` вызывает функцию `pygame.draw.circle`, чтобы нарисовать на экране окружность с центром в переданной точке и радиусом `DrawingTool.PAINT_SIZE`. Это создает эффект рисования на экране.

Наконец, вызывается метод `DrawingTool().start()`, который запускает исполнение программы.

Средство рисования (дополнение)

Решение слишком длинное, чтобы приводить его на страницах книги, но его можно найти в репозитории на GitHub по короткой ссылке *https://rb.gy/tp9qe*.

Ниже приводится его описание.

1. Вначале импортируются необходимые модули: `pygame` для разработки игры, `sys` для получения доступа к системно-зависимым параметрам и функциям и `product` для генерирования декартовых произведений. Библиотека pygame инициализируется методом `pygame.init()`.
2. Определяется класс `DrawingTools` с двумя переменными: `PAINT_BRUSH` и `LINE_TOOL`. Они представляют инструменты рисования, доступные в программе.
3. Определяется класс `ActionButtons` с четырьмя переменными: `CLEAR_CANVAS`, `SAVE_IMAGE`, `PLUS_THICKNESS` и `MINUS_THICKNESS`. Эти переменные представляют доступные в программе кнопки действий.
4. Определяется класс `DrawingTool`, инкапсулирующий функциональность инструмента рисования. В нем определено несколько констант:
   a) `SCREEN_WIDTH` и `SCREEN_HEIGHT` определяют размеры окна рисования;
   b) `SIDEBAR_COLOUR` определяет цвет боковой панели в окне рисования;
   c) `SIDEBAR_BUTTON_COLOUR` определяет цвет кнопок на боковой панели;
   d) `SIDEBAR_WIDTH` определяет ширину боковой панели;
   e) `SIDEBAR_NUM_COLS` определяет количество столбцов на боковой панели;
   f) `SIDEBAR_BUTTONS_RECT_SIZE` определяет размер кнопок на боковой панели;
   g) `THICKNESS_INCREMENT` определяет шаг изменения толщины кисти;
   h) `BACKGROUND_COLOUR` определяет цвет холста рисования;
   i) `BACKGROUND_Y` и `BACKGROUND_X` определяет начальную позицию холста рисования.

5. Конструктор класса `DrawingTool` инициализирует окно рисования и задает его заголовок. Он также инициализирует различные переменные: `is_drawing` (показывает, рисует ли пользователь в данный момент), `is_draw_line_mode` (показывает, включен ли режим рисования линий), `line_initial_point` (начальная точка рисуемой линии), `drawing_colour` (цвет рисования), `drawing_tool` (текущий инструмент рисования), `drawing_snapshot` (моментальный снимок холста рисования) и `thickness` (текущая толщина кисти).
6. Метод `start` содержит главный цикл программы. Сначала он вызывает метод `clear_canvas`, чтобы очистить холст. Затем в цикле обрабатываются события, инициированные пользователем (например, щелчки мышью и закрытие окна).
   a. Если пользователь нажимает кнопку закрытия окна, то вызывается метод `pygame.quit()`, завершающий игру, и метод `sys.exit()`, который осуществляет выход из программы.
   b. Если пользователь нажимает кнопку мыши, то вызывается метод `handle_mouse_button_down`.
   c. Если пользователь отпускает кнопку мыши, то вызывается метод `handle_mouse_button_up`.
7. Метод `render_side_bar method` отвечает за отрисовку боковой панели в окне инструмента рисования. Он создает словарь `clickables`, в котором хранятся чувствительные к щелчку прямоугольники. Затем он закрашивает боковую панель цветом фона, рисует прямоугольники цветов, кнопки действий и селектор толщины. Метод возвращает словарь `clickables`.
8. Метод `render_sidebar_colour_options` рисует прямоугольники цветов на боковой панели. Он пользуется функцией `product` из модуля `itertools` для генерирования всех возможных комбинаций RGB-значений (0 и 255). Затем он обходит цвета и рисует прямоугольники для каждого цвета. Эти прямоугольники добавляются в словарь `clickables`.
9. Метод `render_sidebar_action_buttons` рисует кнопки действий на боковой панели. Он использует список предопределенных инструментов и кнопок и обходит их, чтобы нарисовать прямоугольники и текст каждой кнопки. Эти прямоугольники добавляются в словарь `clickables`.
10. Метод `render_thickness_selector` рисует селектор толщины на боковой панели. Он рисует кнопки с изображением плюса

и минуса для увеличения и уменьшения толщины соответственно. Кроме того, рисуется текущая толщина. Кнопки добавляются в словарь `clickables`.

11. Метод `handle_mouse_button_down` вызывается в ответ на нажатие кнопки мыши. Если курсор мыши в этот момент указывал на фон холста, то проверяется текущий инструмент рисования. Если это кисть, то переменной `is_drawing` присваивается значение `True`, это означает, что пользователь рисует. Если же текущим является инструмент рисования линий, то переменная `is_draw_line_mode` меняет значение на противоположное. Если она стала равна `True`, то создается моментальный снимок текущего состояния холста и сохраняется начальное положение мыши для рисования линии.
12. В ответ на отпускание кнопки мыши вызывается метод `handle_mouse_button_up`. Если курсор мыши в этот момент указывал на фон холста, то проверяется текущий инструмент рисования. Если это кисть, то переменной `is_drawing` присваивается значение `False`, это означает, что пользователь прекратил рисовать. Если же курсор мыши в момент щелчка не указывал на холст, то вызывается метод `action_sidebar_btn_pressed`.
13. Метод `action_sidebar_btn_pressed` вызывается, когда нажата кнопка действия или прямоугольник цвета на боковой панели. Он проверяет, что именно было нажато, и выполняет соответствующее действие. Если был нажат прямоугольник цвета, то обновляется текущий цвет рисования. В случае кнопок действия метод может обновить текущий инструмент рисования, очистить холст, сохранить изображение или модифицировать толщину.
14. Метод `clear_canvas` очищает холст, рисуя прямоугольник заданным цветом фона.
15. Метод `save_image` сохраняет текущий рисунок в виде изображения. Он создает подповерхность на экране, используя фоновый прямоугольник в качестве подлежащей сохранению области. Если переменная `save_to_disk` равна `True`, то подповерхность сохраняется в файле изображения. Метод возвращает подповерхность.
16. Метод `draw` отвечает за рисование на экране окружностей с заданным центром линией заданной толщины и цвета.

17. Метод `draw_line` отвечает за рисование на экране прямой линии, соединяющей начальное положение мыши с ее текущим положением. Он также рисует окружности в конечных точках линии, чтобы создать эффект закругления.

Наконец, метод `DrawingTool().start()` запускает исполнение программы.

Мэттью Уайтсайд

Python в задачах и упражнениях

Особенности этой книги:

- более 60 задач, каждая из которых раскрывает новую грань программирования на Python, расширяет ваши познания и позволяет уверенно подойти к реальной практике;
- постепенно усложняющиеся серьезные и шуточные задачи предлагают идеальный способ изучить Python, отточить навыки работы с ним и познакомиться с библиотеками, о которых вы раньше не знали;
- во всех задачах используются самые актуальные версии и библиотеки, книга ориентируется на Python 3 и идет в ногу с изменениями, которые могут повлиять на решения;
- помощь эксперта помогает преодолеть препятствия – автор приводит указания и решения всех задач;